

Demand-aware and distributed cloud networking: Let's get physical!

Stefan Schmid (University of Vienna)



DCC @ DISC 2019
Budapest, Hungary

Trend: Data-Centric Applications



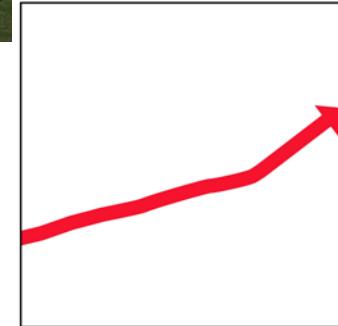
Trend: Data-Centric Applications

Datacenters (“hyper-scale”)



Trend: Data-Centric Applications

Datacenters (“hyper-scale”)



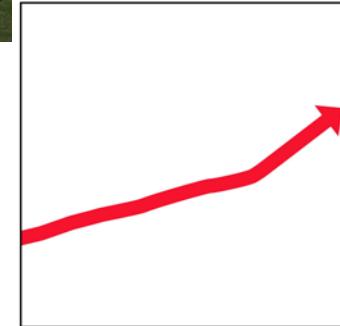
Source: Facebook

Trend: Data-Centric Applications

Datacenters ("hyper-scale")



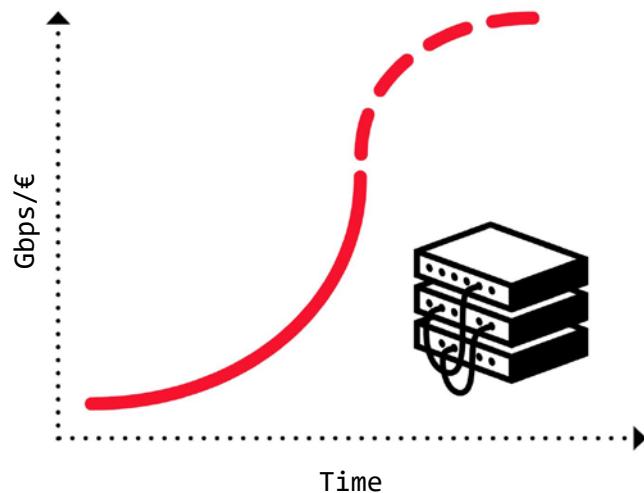
Interconnecting networks:
a **critical infrastructure**
of our digital society.



Source: Facebook

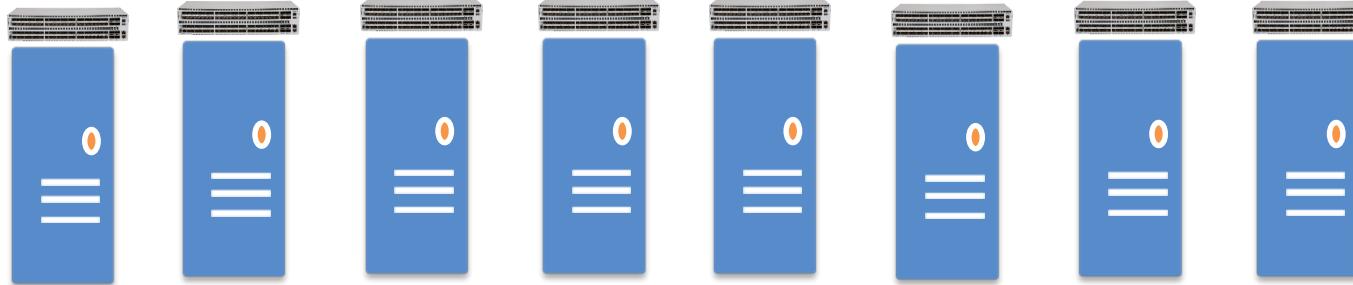
Problem: Huge Infrastructure, Inefficient Cuse

- Network equipment reaching capacity limits
 - Transistor density rates stalling
 - “End of **Moore’s Law** in networking” [1]
- Hence: more equipment, larger networks
- Resource intensive and: **inefficient**

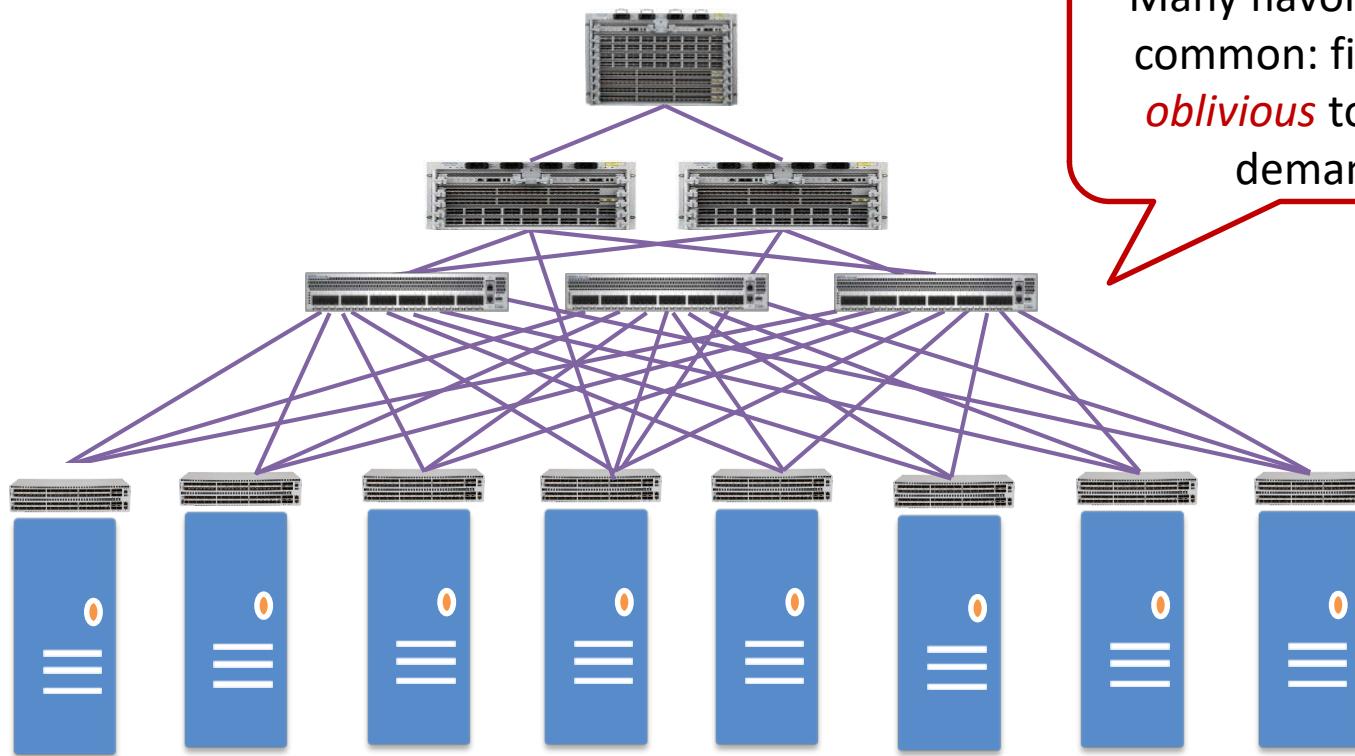


Root Cause: Fixed and Demand-Oblivious Topology

How to interconnect?



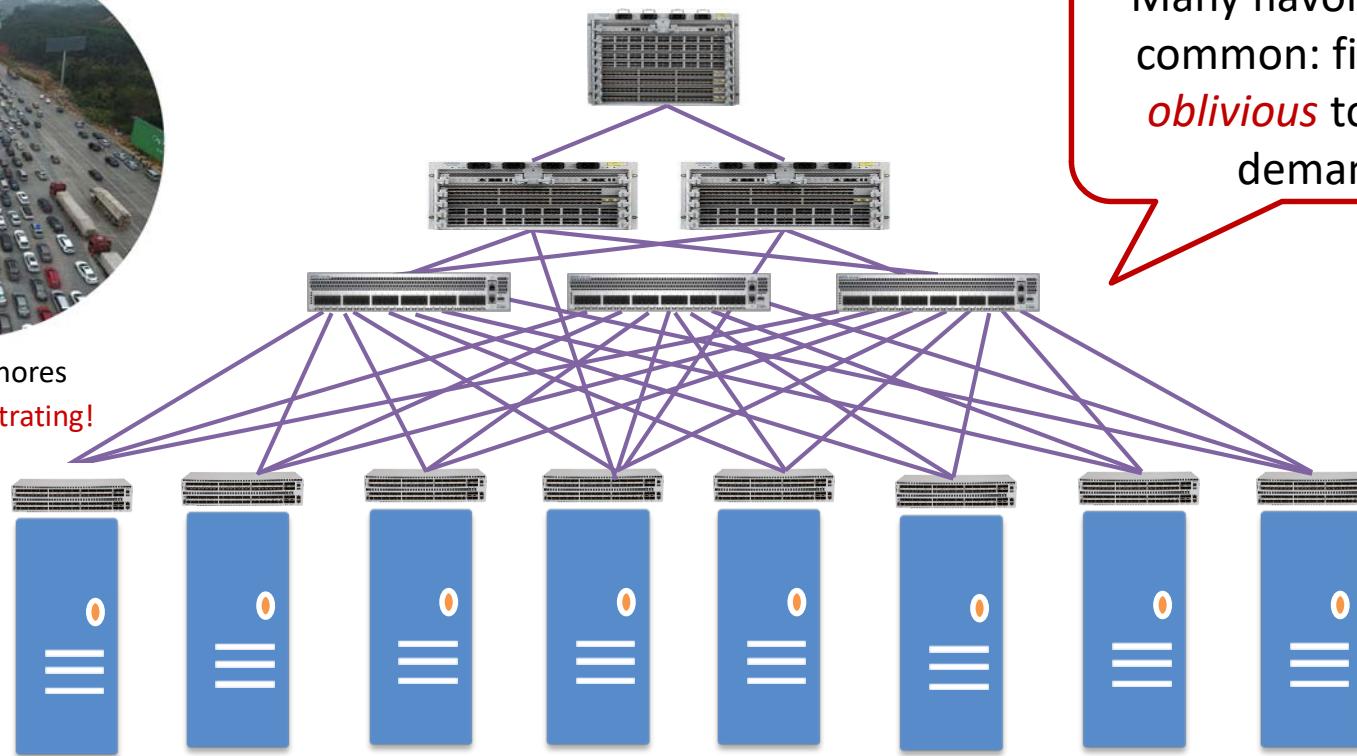
Root Cause: Fixed and Demand-Oblivious Topology



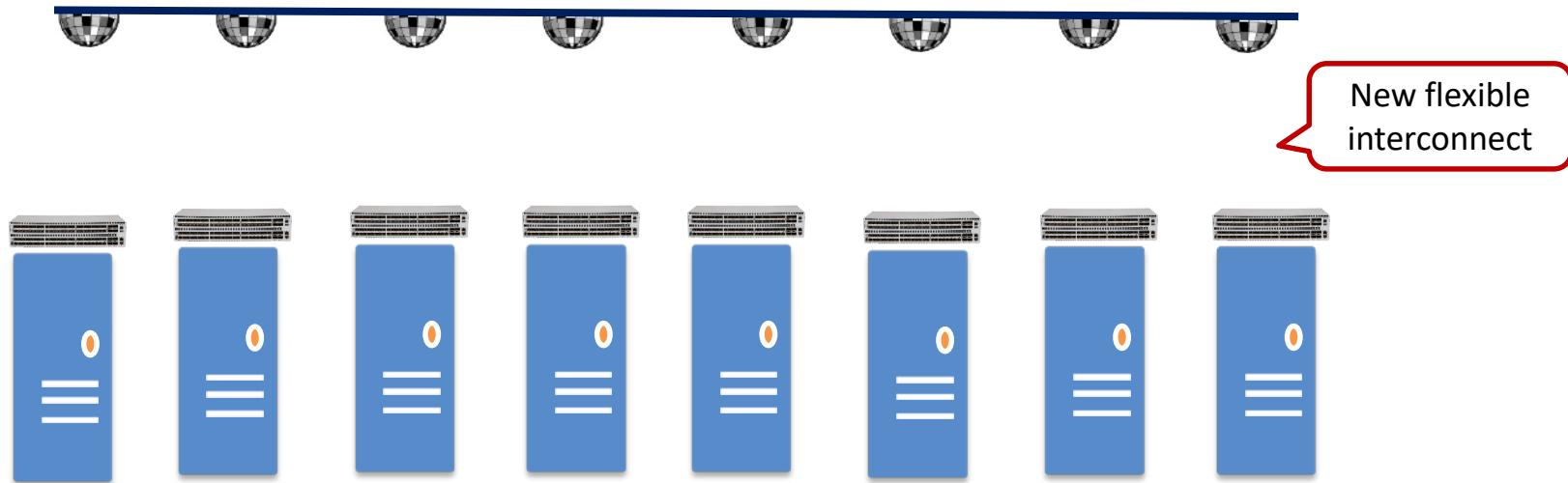
Root Cause: Fixed and Demand-Oblivious Topology



Highway which ignores
actual traffic: **frustrating!**



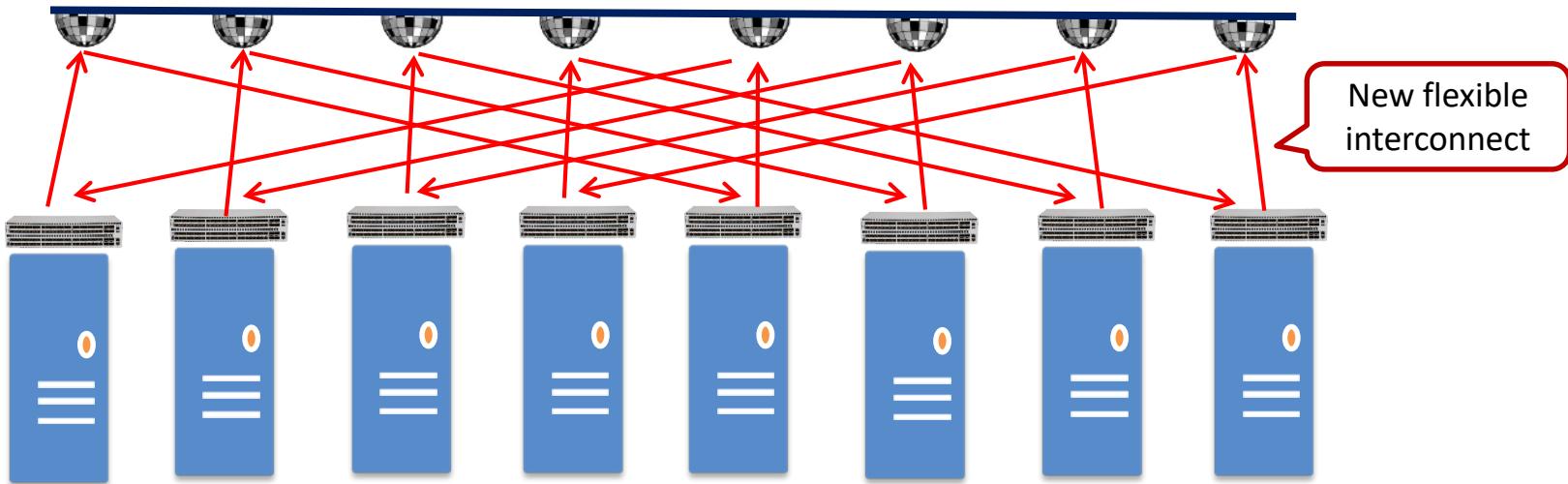
Our Vision: Self-Adjusting Networks



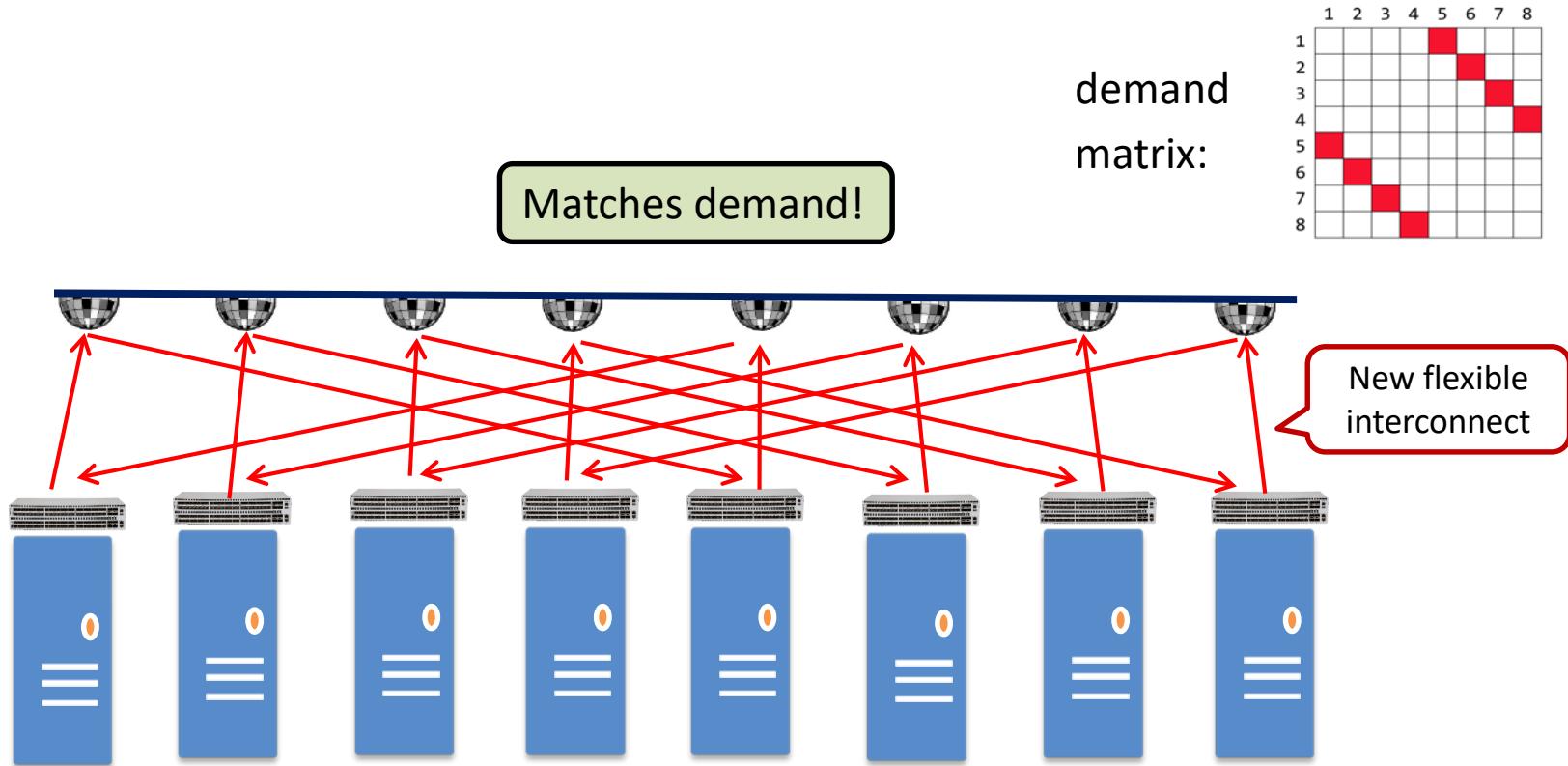
Our Vision: Self-Adjusting Networks

demand
matrix:

1	2	3	4	5	6	7	8
1							
2							
3							
4							
5							
6							
7							
8							



Our Vision: Self-Adjusting Networks



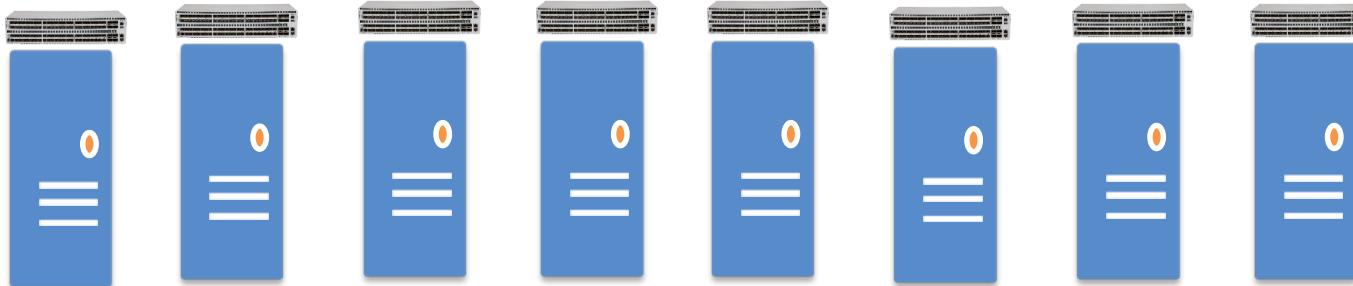
Our Vision: Self-Adjusting Networks

demand
matrix:

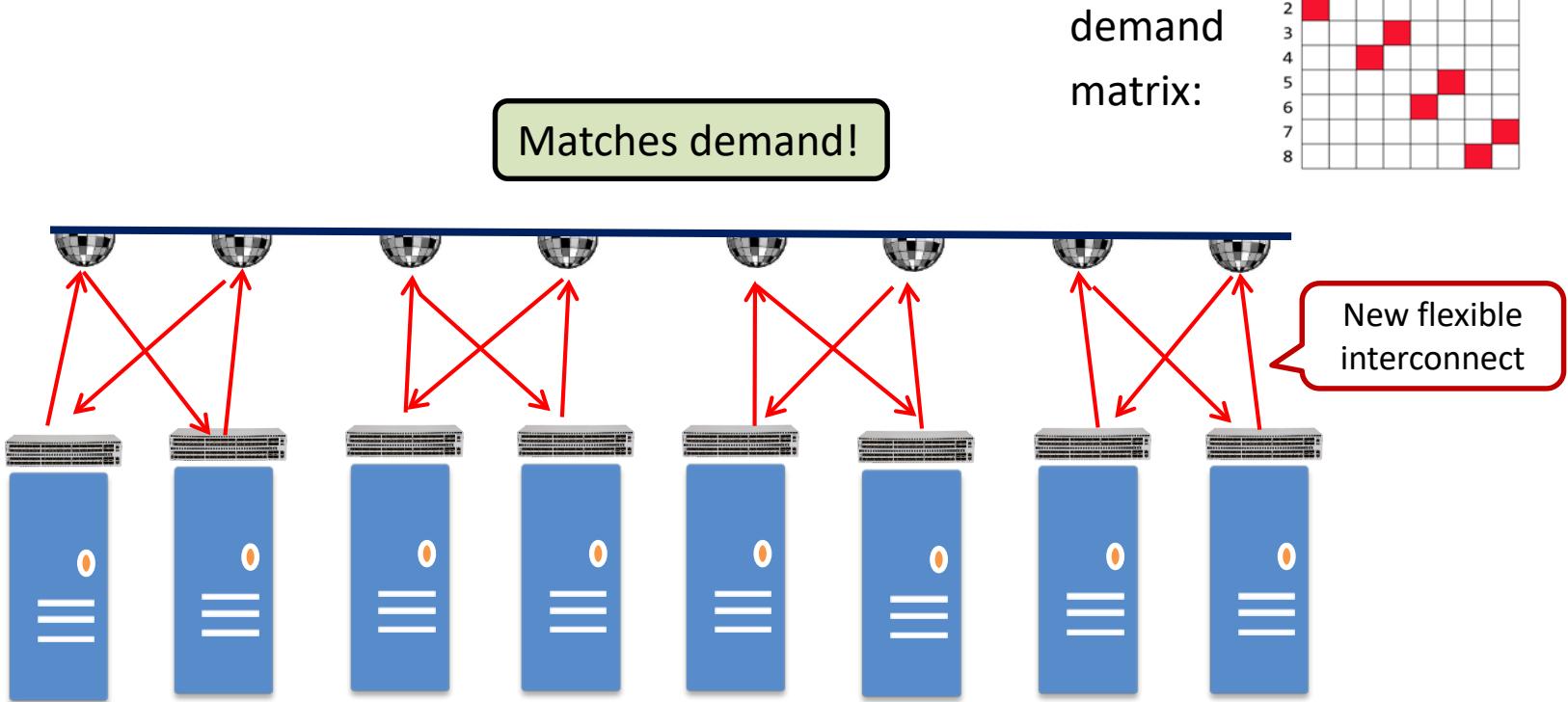
1	2	3	4	5	6	7	8
2							
3							
4							
5							
6							
7							
8							



New flexible
interconnect



Our Vision: Self-Adjusting Networks



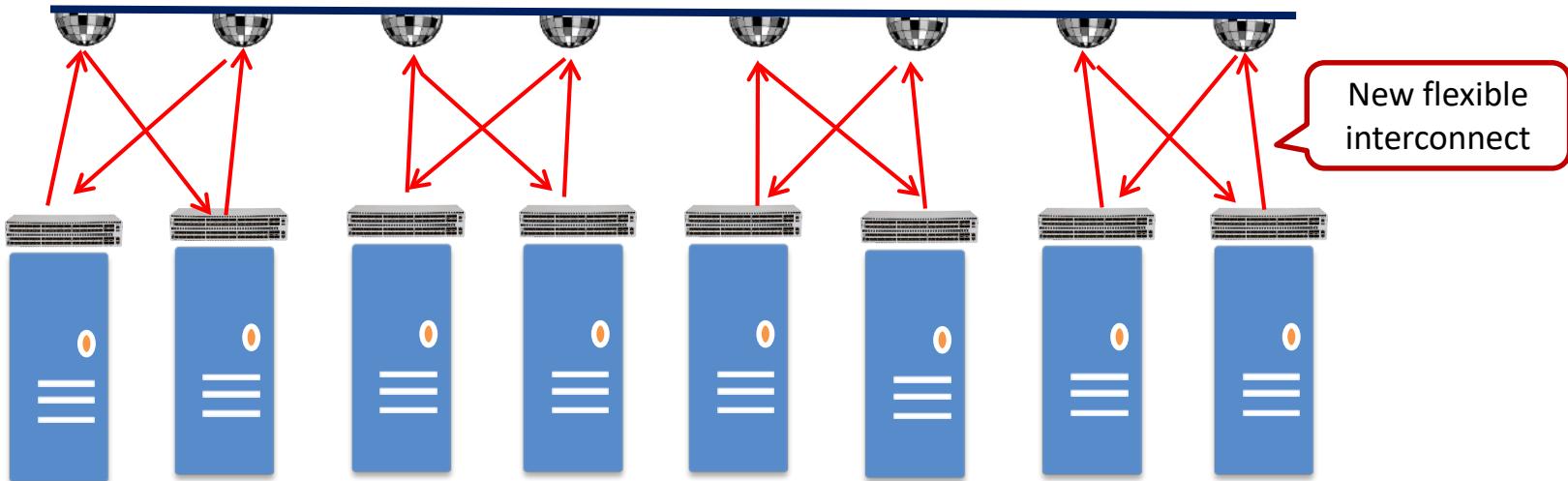
Our Vision: Self-Adjusting Networks



Self-adjusting
networks!

demand
matrix:

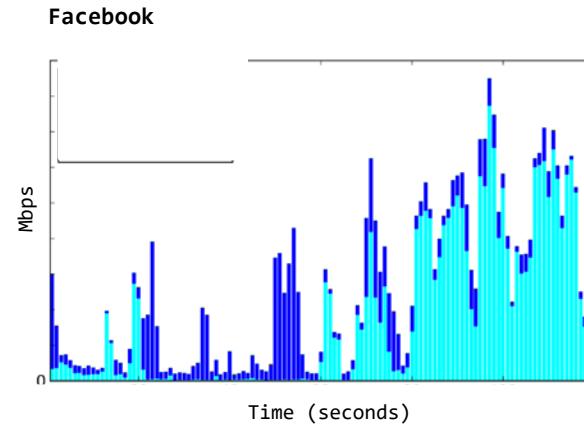
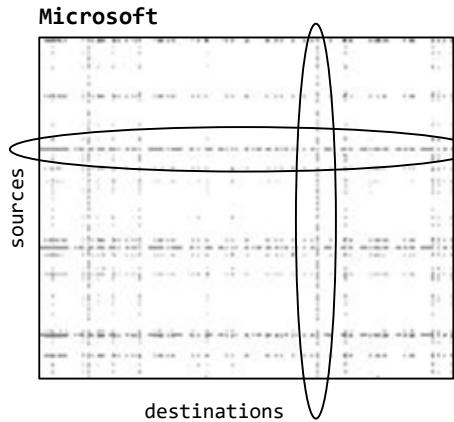
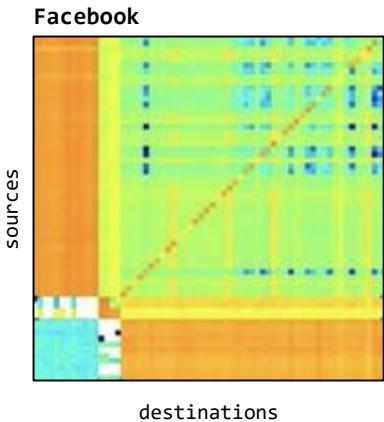
	1	2	3	4	5	6	7	8
1	1							
2		1						
3			1					
4				1				
5					1			
6						1		
7							1	
8								1



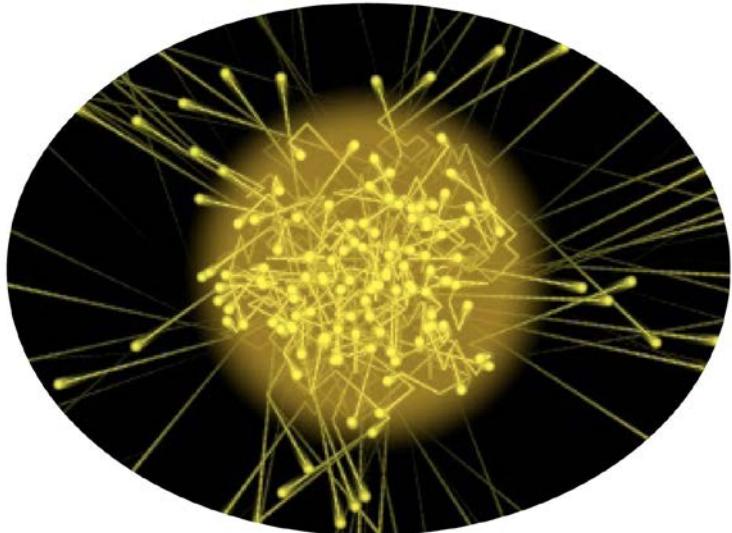
Empirical Motivation

Empirical studies:

traffic matrices **sparse** and **skewed**

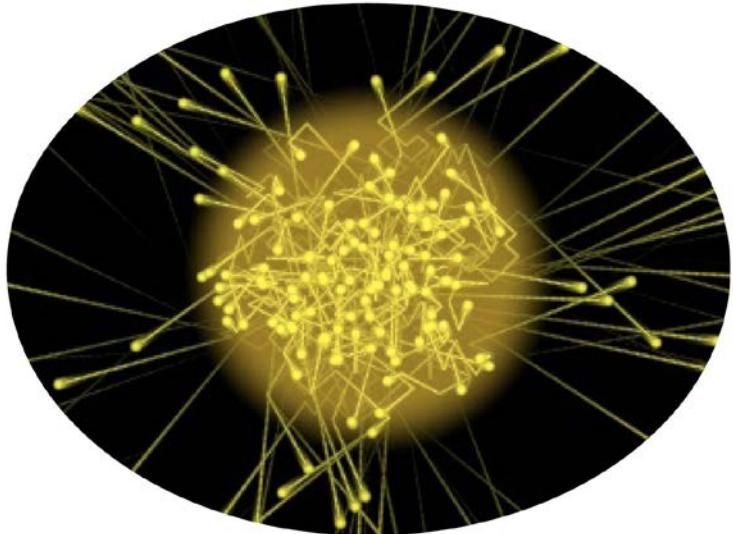


Emerging Enabling Technologies



Photonics

Emerging Enabling Technologies



Photonics

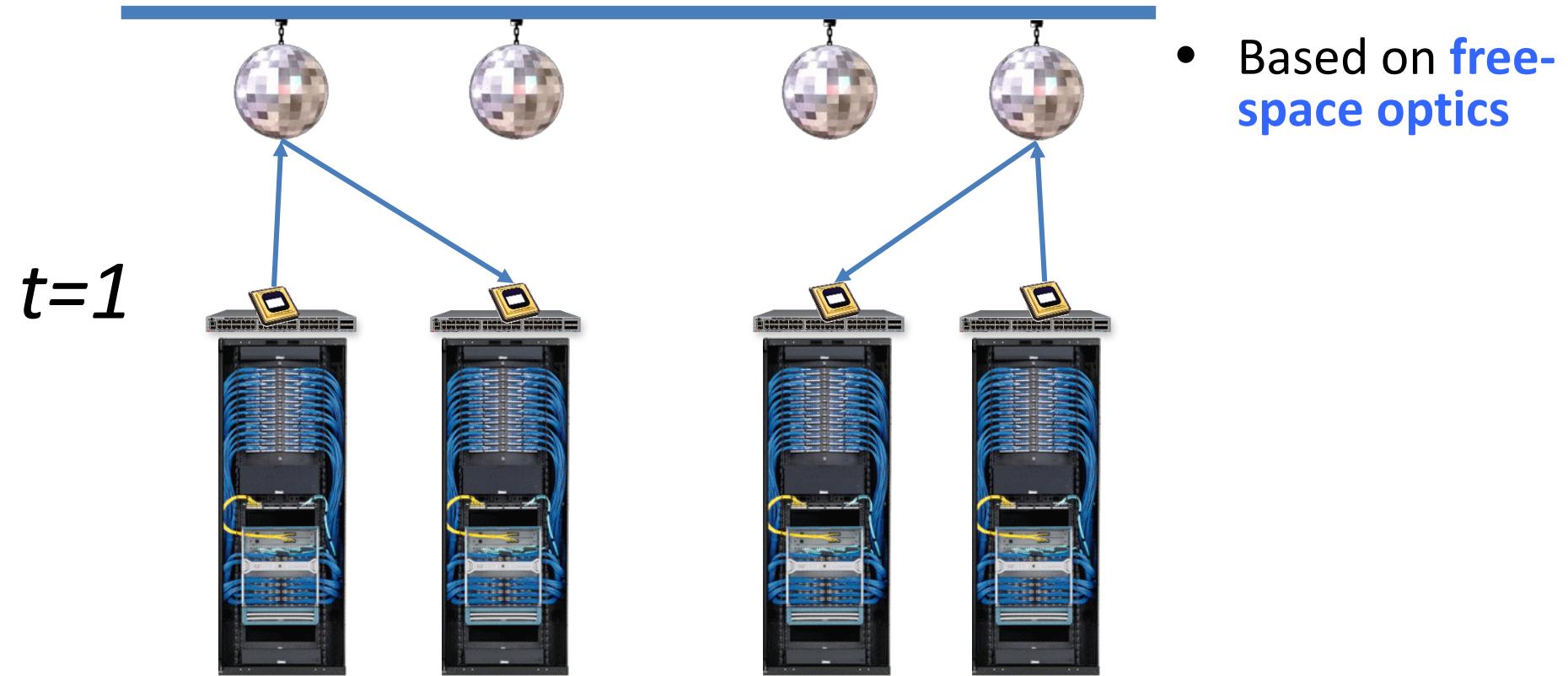
- US National Research Council: “photons are *the new electrons*”
- H2020: “Photonics one of only five *key enabling technologies* for future prosperity”

Example: ProjecToR

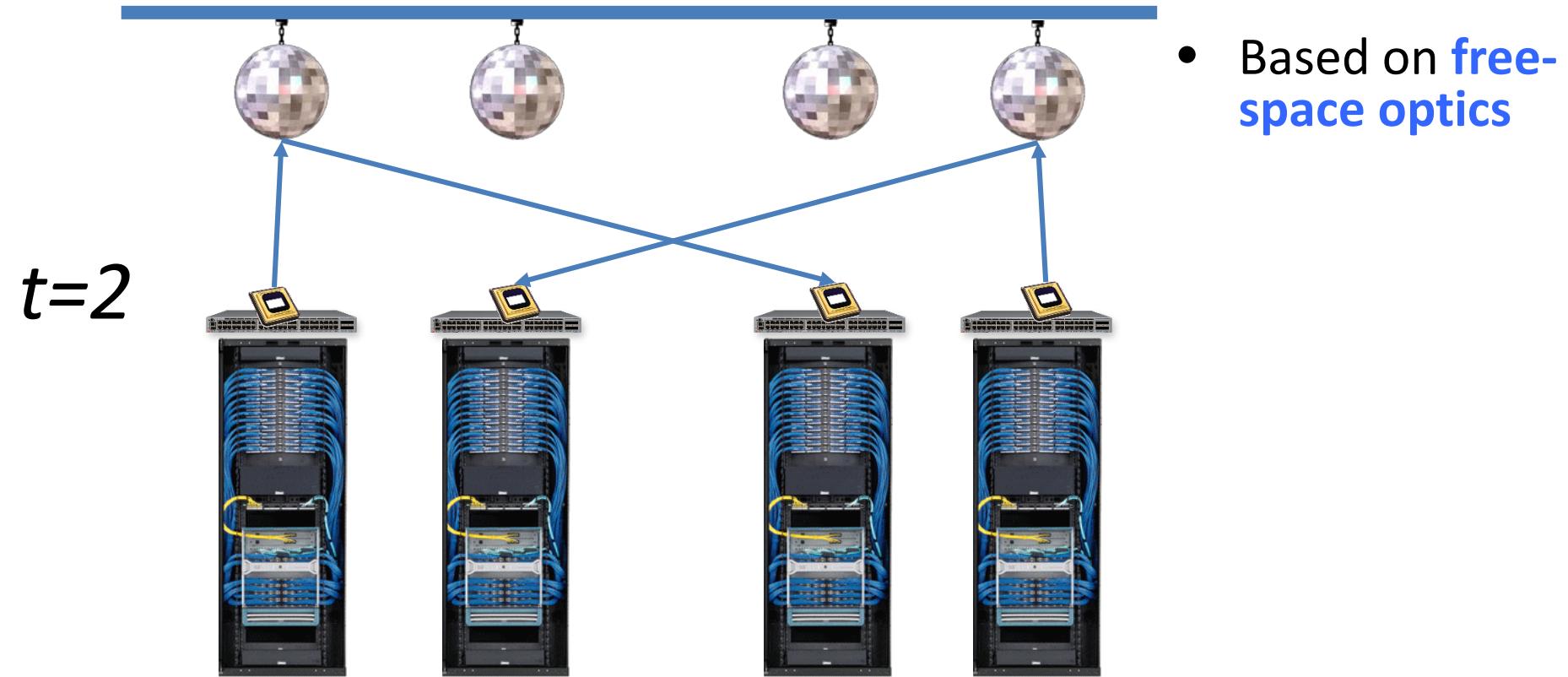


Example: Manya Ghobadi et al.
Kudos for some slides!

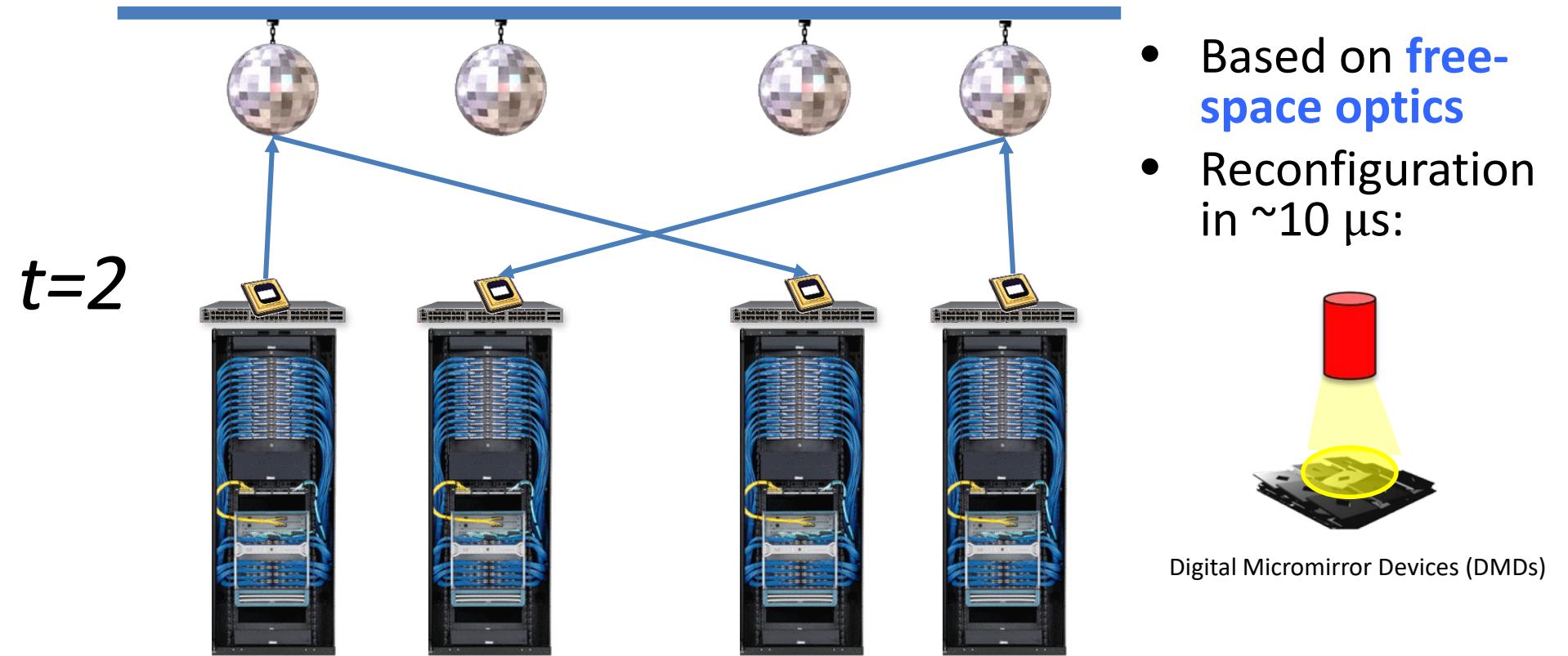
ProjectToR



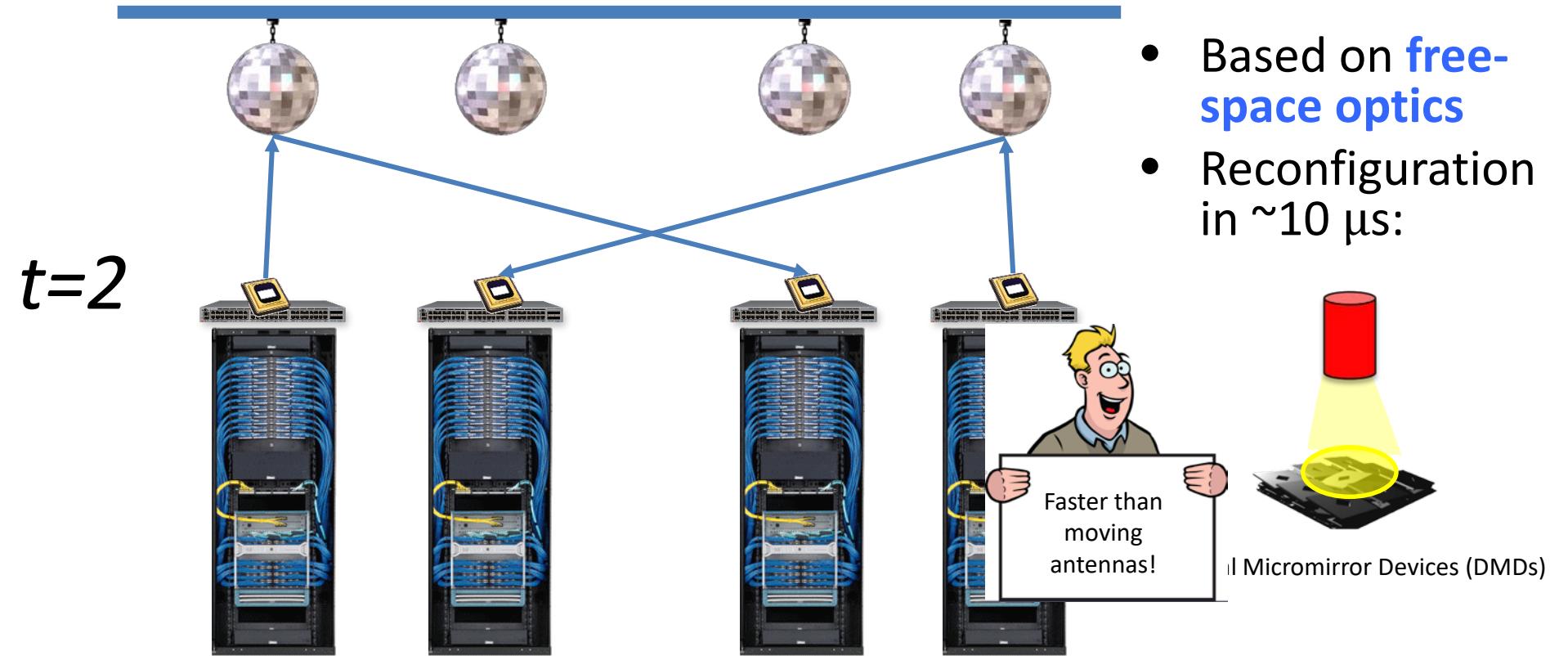
ProjectToR



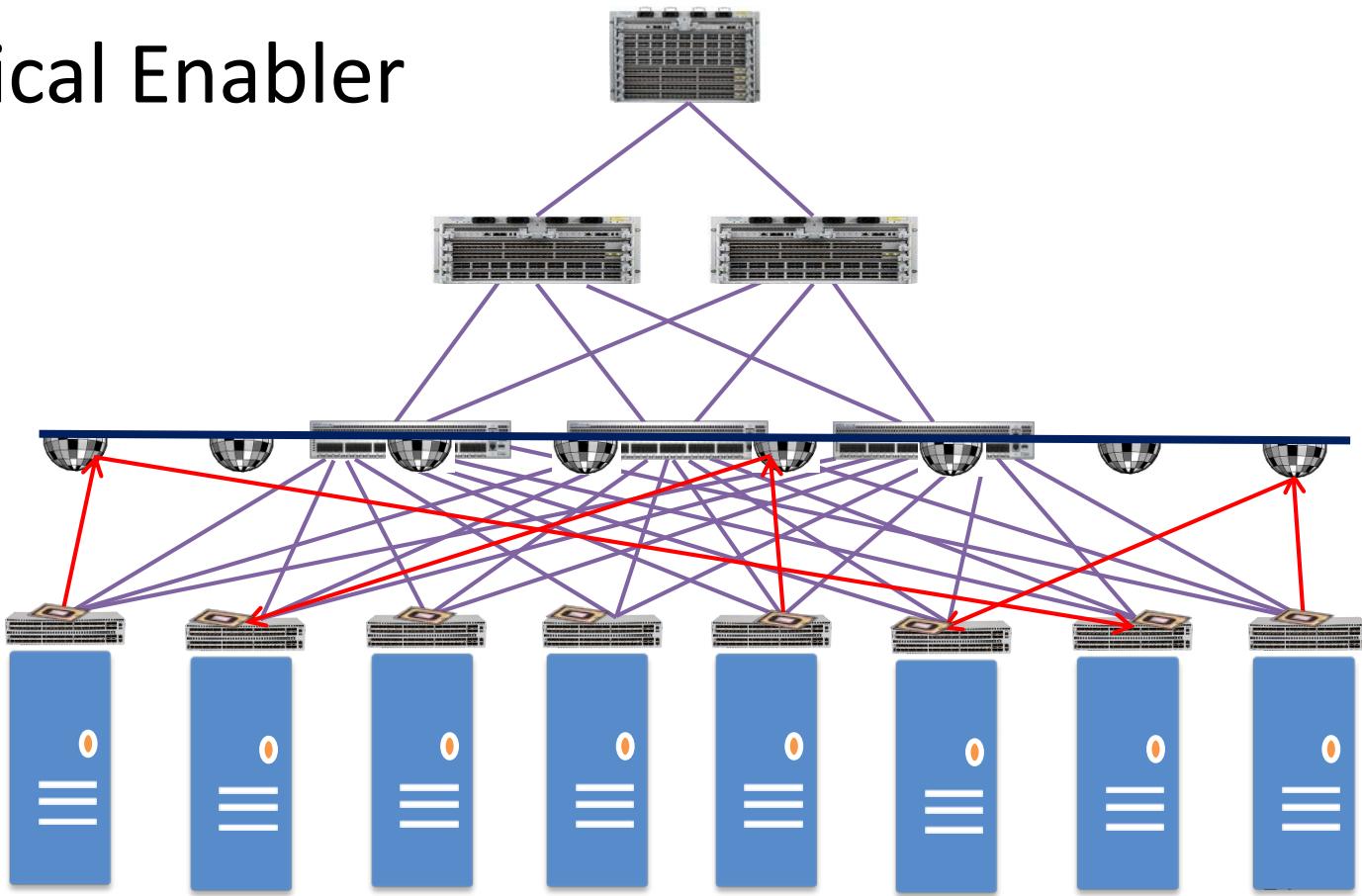
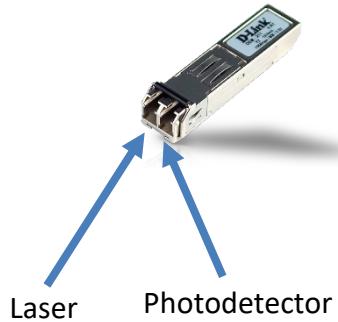
ProjectToR



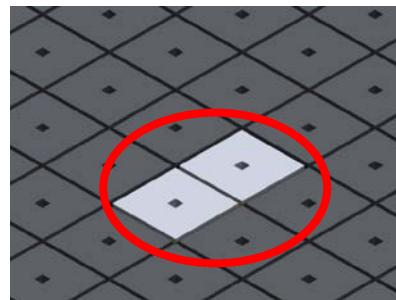
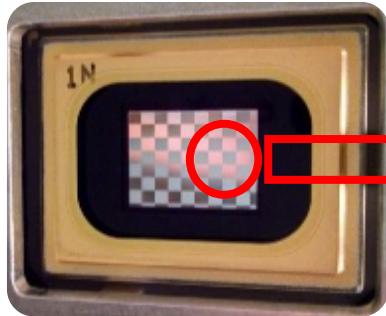
ProjectToR



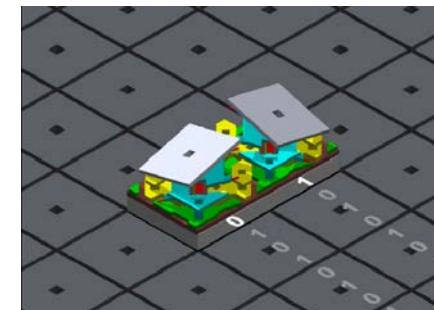
ProjectToR in More Details: Technological Enabler



ProjecToR in More Details: DMDs



Array of
micromirrors

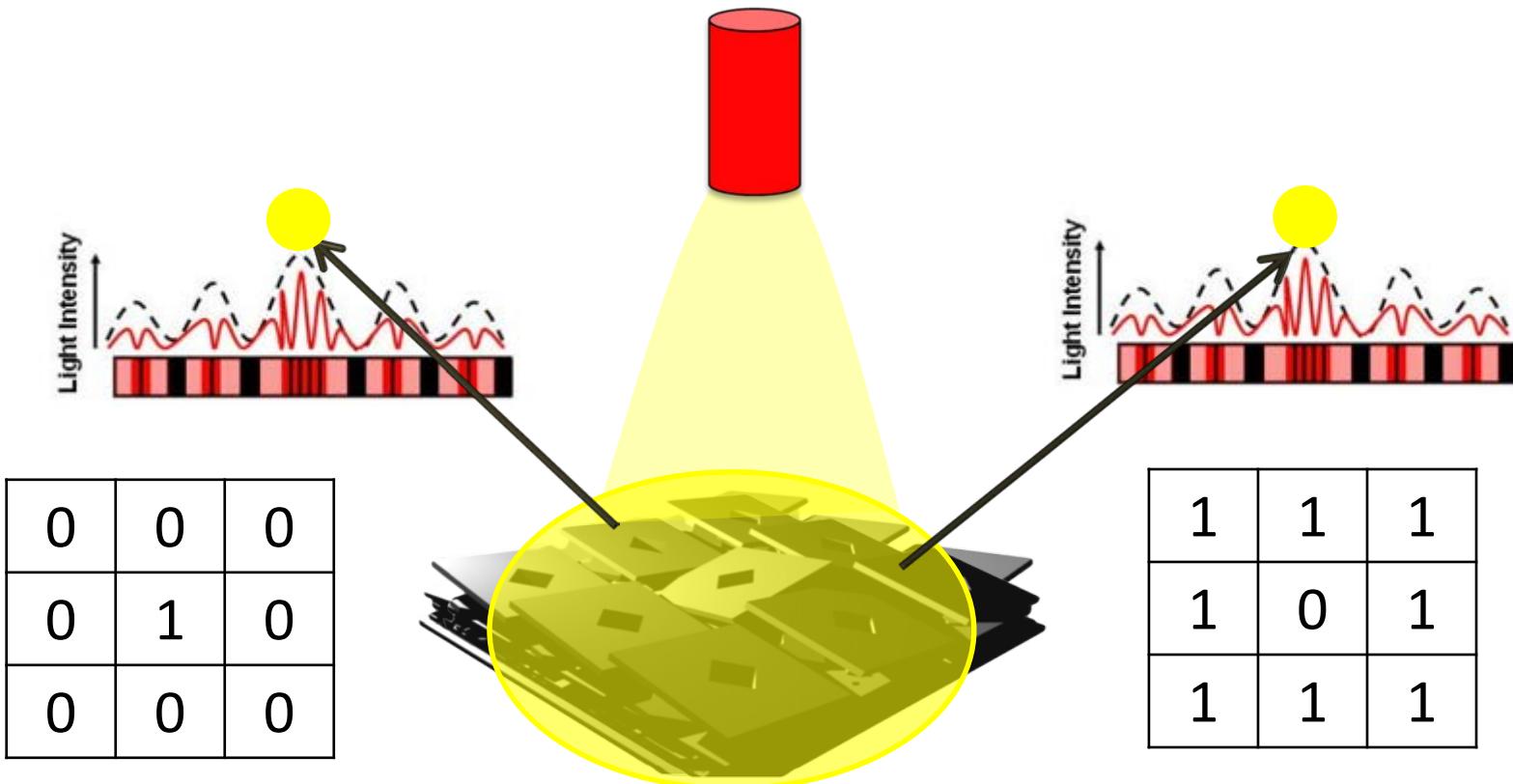


Memory cell

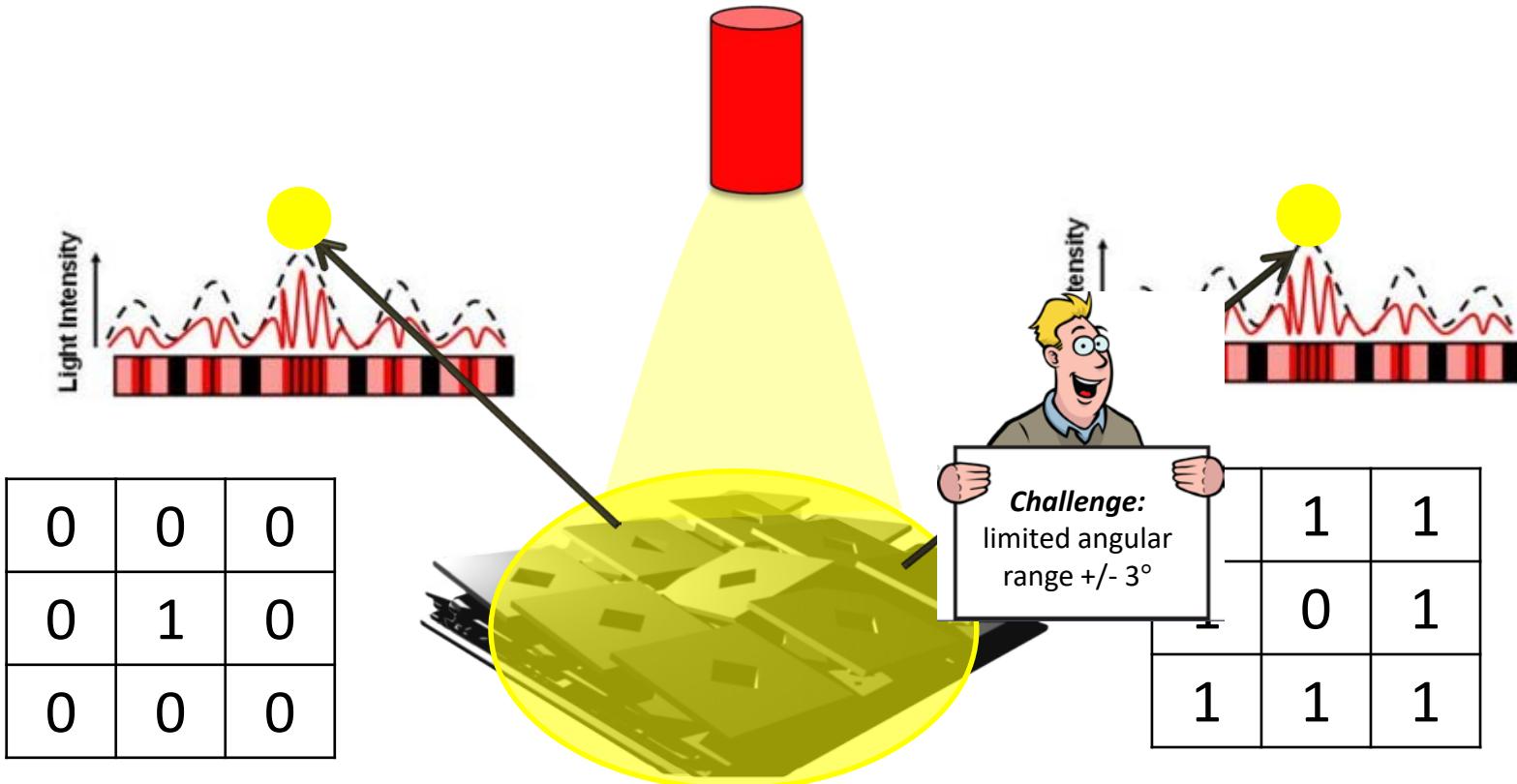


- Each micromirror can be turned on/off
- Essentially a **0/1-image**: e.g., array size 768 x 1024
- Direction of the diffracted light can be finely tuned

ProjectToR in More Details: DMDs to Redirect Light *Fast*



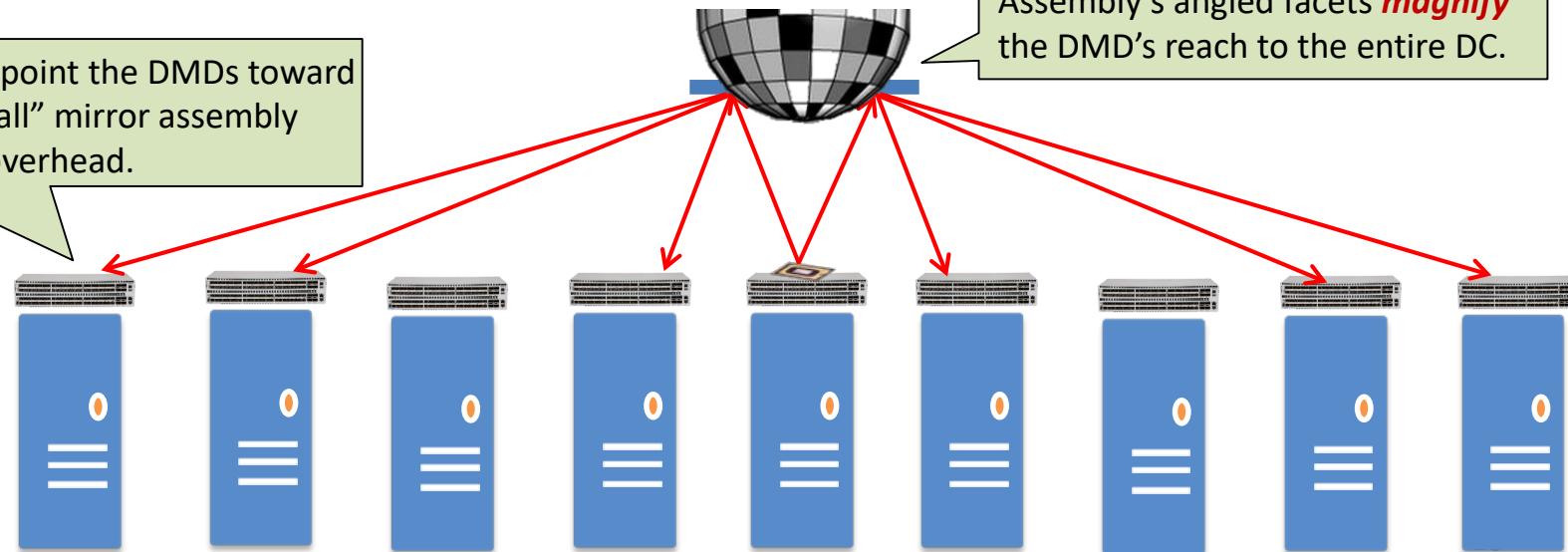
ProjectToR in More Details: DMDs to Redirect Light *Fast*



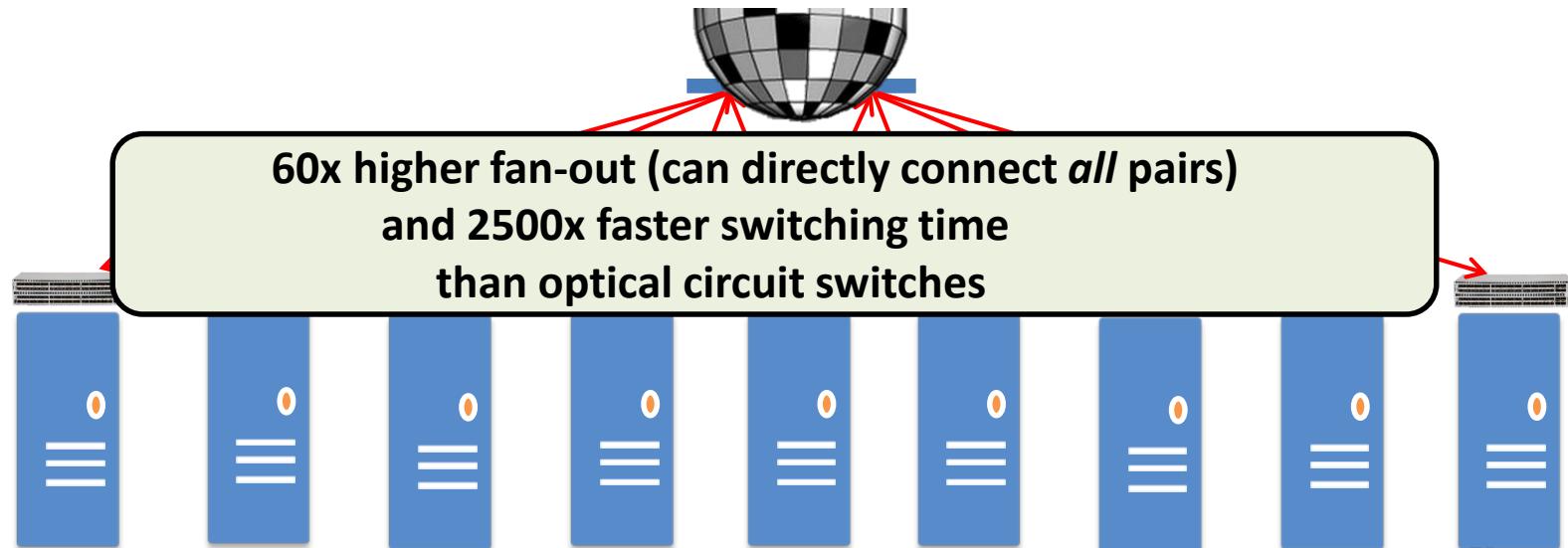
ProjectToR in More Details: Coupling DMDs with angled mirrors

Coupling: point the DMDs toward a “disco-ball” mirror assembly installed overhead.

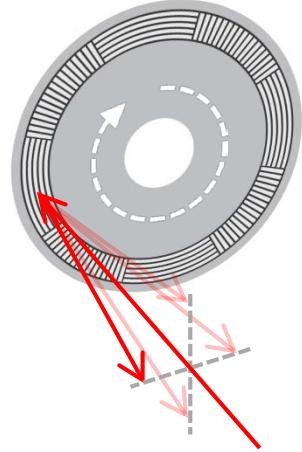
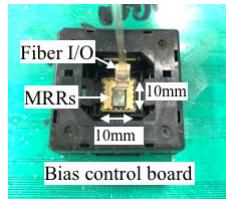
Assembly's angled facets **magnify** the DMD's reach to the entire DC.



ProjecToR in More Details: Coupling DMDs with angled mirrors



Other Technologies



Based on silicon photonics

2-NEMS

Rotating disks

Further reading:

Wade et al., A Bandwidth-Dense, Low Power Electronic-Photonic Platform and Architecture for Multi-Tbps Optical I/O [OFC'18]

Porter et al., "Integrating Microsecond Circuit Switching into the Data Center", Sigcomm'13

Timeline

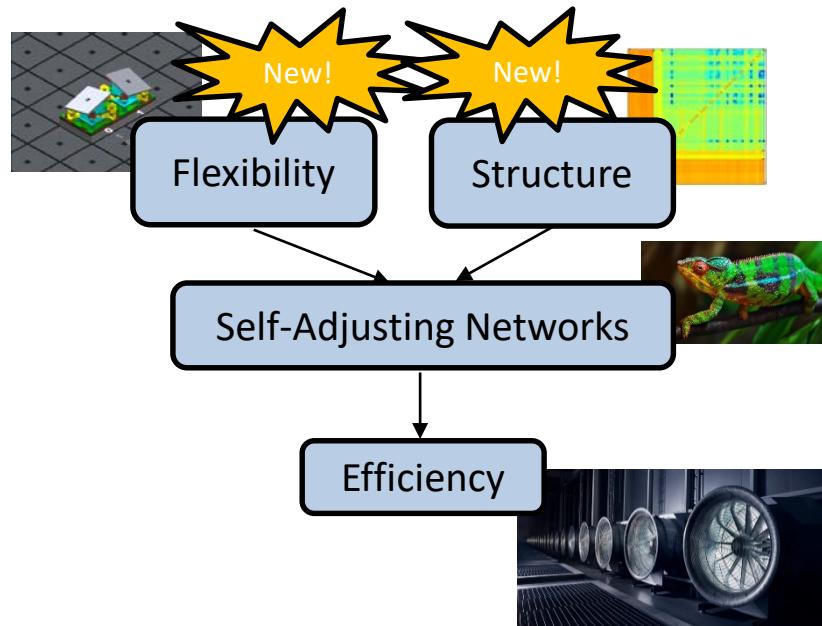
Reconfiguration time: from milliseconds **to microseconds** (and decentralized).

Survey of Reconfigurable Data Center Networks. Foerster and Schmid.
SIGACT News, 2019.

-
- | | |
|------|---|
| 2009 | - <i>Flyways</i> [51]: Steerable antennas (narrow beamwidth at 60 GHz [78]) to serve hotspots |
| 2010 | - <i>Helios</i> [33]/ <i>c-Through</i> [98, 99]: Hybrid switch architecture, maximum matching (Edmond's algorithm [30]), single-hop reconfigurable connections ($O(10)ms$ reconfiguration time).
- <i>Proteus</i> [21, 89]: k reconfigurable connections per ToR, multi-hop path stitching, multi-hop reconfigurable connections (weighted b -matching [69], edge-exchanges for connectivity [72], wavelength assignment via edge-coloring [67] on multigraphs) |
| 2011 | - Extension of <i>Flyways</i> [51] to better handle practical concerns such as stability and interference for 60GHz links, along with greedy heuristics for dynamic link placement [45] |
| 2012 | - <i>Mirror Mirror on the ceiling</i> [106]: 3D-beamforming (60 Ghz wireless), signals bounce off the ceiling |
| 2013 | - <i>Mordia</i> [31, 32, 77]: Traffic matrix scheduling, matrix decomposition (Birkhoff-von-Neumann (BvN) [18, 97]), fiber ring structure with wavelengths ($O(10)\mu s$ reconfiguration time)
- <i>SplayNets</i> [6, 76, 82]: Fine-grained and online reconfigurations in the spirit of self-adjusting datastructures (all links are reconfigurable), aiming to strike a balance between short route lengths and reconfiguration costs |
| 2014 | - <i>REACToR</i> [56]: Buffer burst of packets at end-hosts until circuit provisioned, employs [77]
- <i>Firefly</i> [14]: Combination of Free Space Optics and Galvo/switchable mirrors (small fan-out) |
| 2015 | - <i>Solstice</i> [57]: Greedy perfect matching based hybrid scheduling heuristic that outperforms BvN [77]
- Designs for optical switches with a reconfiguration latency of $O(10)ns$ [3] |
| 2016 | - <i>ProjectToR</i> [39]: Distributed Free Space Optics with digital micromirrors (high fan-out) [38] (Stable Matching [26]), goal of (starvation-free) low latency
- <i>Eclipse</i> [95, 96]: $(1 - 1/e^{(1-\varepsilon)})$ -approximation for throughput in traffic matrix scheduling (single-hop reconfigurable connections, hybrid switch architecture), outperforms heuristics in [57] |
| 2017 | - <i>DAN</i> [7, 8, 11, 12]: Demand-aware networks based on reconfigurable links only and optimized for a demand snapshot, to minimized average route length and/or minimize load
- <i>MegaSwitch</i> [23]: Non-blocking circuits over multiple fiber rings (stacking rings in [77] doesn't suffice)
- <i>Rotornet</i> [63]: Oblivious cyclical reconfiguration w. selector switches [64] (Valiant load balancing [94])
- <i>Tale of Two Topologies</i> [105]: Convert locally between Clos [24] topology and random graphs [87, 88] |
| 2018 | - <i>DeepConf</i> [81]/ <i>xWeaver</i> [102]: Machine learning approaches for topology reconfiguration |
| 2019 | - Complexity classifications for weighted average path lengths in reconfigurable topologies [34, 35, 36]
- <i>ReNet</i> [13] and <i>Push-Down-Trees</i> [9] providing statically and dynamically optimal reconfigurations
- <i>DisSplayNets</i> [75]: fully decentralized <i>SplayNets</i>
- <i>Opera</i> [60]: Maintaining expander-based topologies under (oblivious) reconfiguration |

Our Vision

Our Vision



How much can we profit?

A Simple Answer

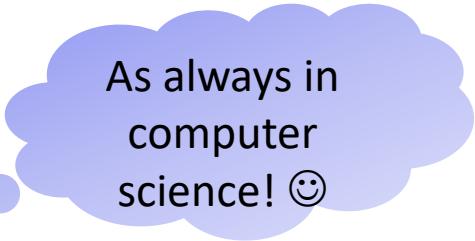
Demand-Oblivious Networks =



The *CS* Answer

- It depends...

The CS Answer



As always in
computer
science! 😊

- It depends...

The CS Answer

As always in
computer
science! ☺

- It depends...
- ... on the demand!



We need **metrics!**

Roadmap

- Entropy: A metric for demand-aware networks?
 - Intuition
 - A lower bound
 - Algorithms achieving entropy bounds
- From static to dynamic demand-aware networks
 - Empirical motivation
 - A connection to self-adjusting datastructures



Roadmap

- Entropy: A metric for demand-aware networks?
 - Intuition
 - A lower bound
 - Algorithms achieving entropy bounds
- From static to dynamic demand-aware networks
 - Empirical motivation
 - A connection to self-adjusting datastructures



A Simple Example

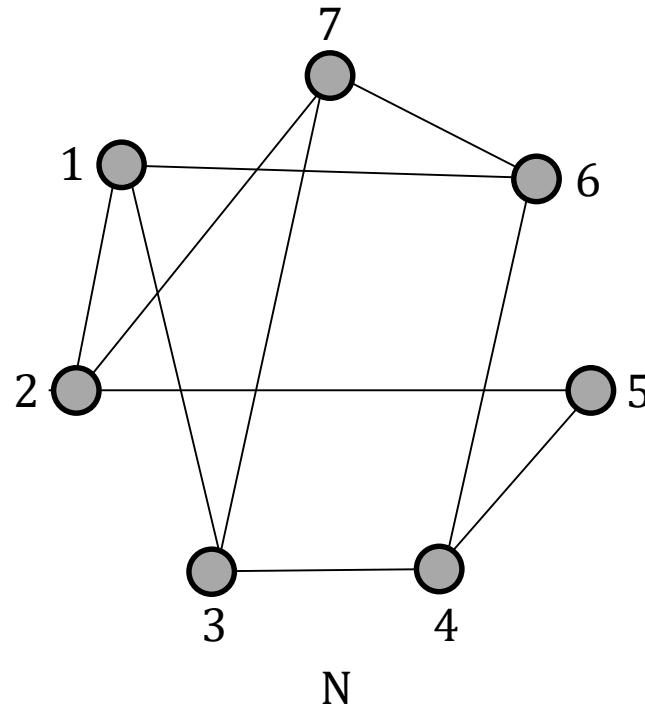
Input: Workload

Destinations

		Destinations						
		1	2	3	4	5	6	7
Sources	1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
	2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
	3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
	4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
	5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
	6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
	7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

\mathcal{D}

Output: Constant-Degree DAN



N

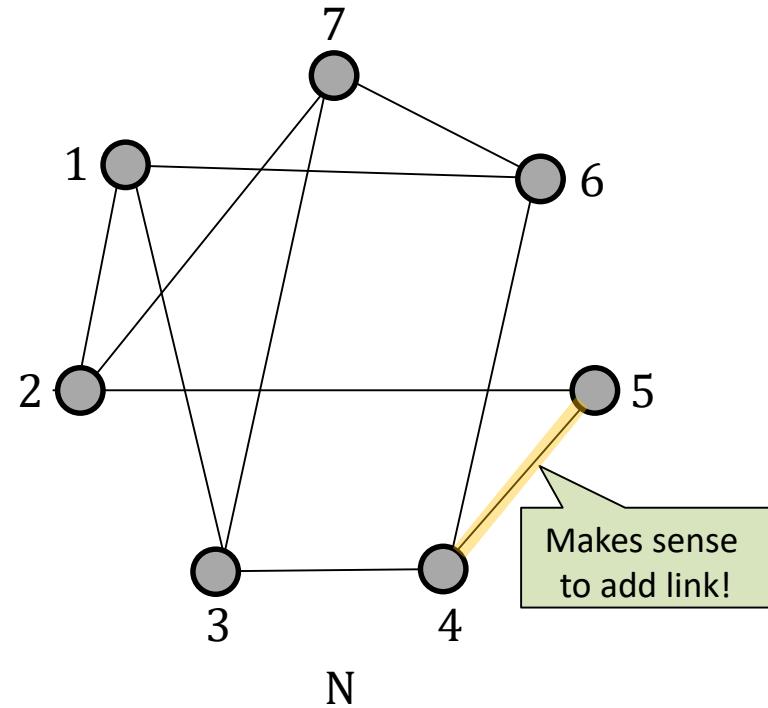
Input: Workload

Destinations

	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	Much from 4 to 5.		
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

\mathcal{D}

Output: Constant-Degree DAN



Input: Workload

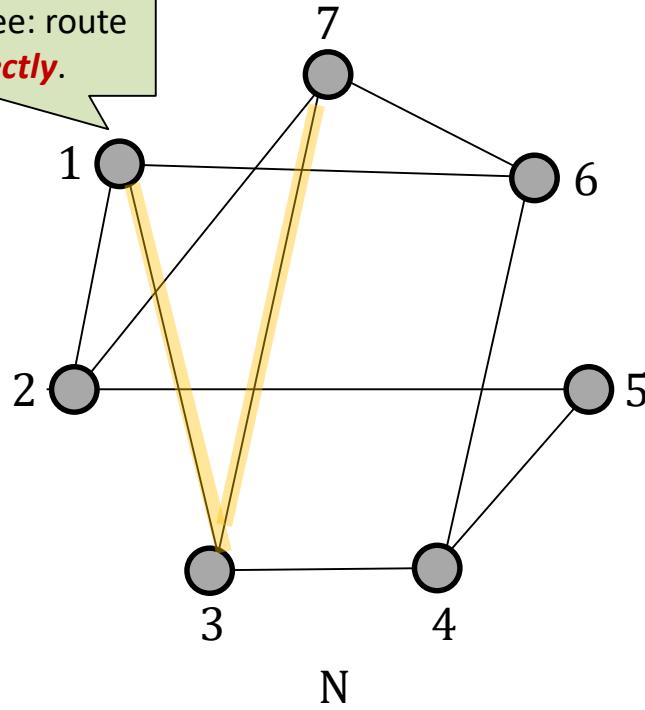
Output: Constant-Degree DAN

1 communicates to many.

		Destinations						
		2	3	4	5	6	7	
Sources	1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
	2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
	3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
	4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
	5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
	6	$\frac{2}{65}$	0	0	0	0	$\frac{3}{65}$	
	7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

\mathcal{D}

Bounded degree: route to 7 *indirectly*.



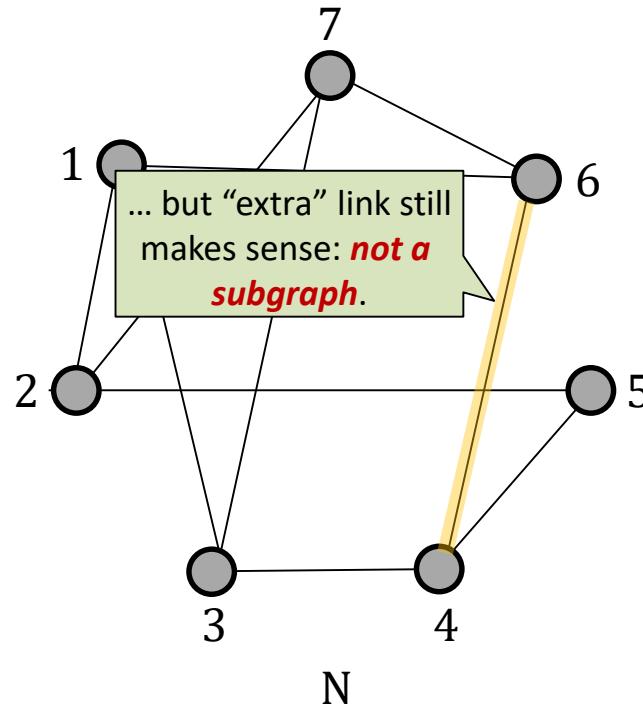
Input: Workload

Destinations

	1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0		
6	$\frac{2}{65}$	0	0	0	0		$\frac{1}{65}$
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

\mathcal{D}

Output: Constant-Degree DAN



Objective: Expected Route Length

$$\text{ERL}(\mathcal{D}, N) = \sum_{(u,v) \in \mathcal{D}} p(u, v) \cdot d_N(u, v)$$

DAN N of degree Δ

path length on N

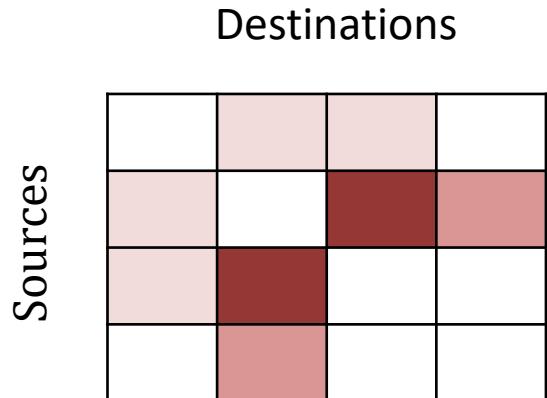
$\mathcal{D}[p(i,j)]$: joint distribution

frequency

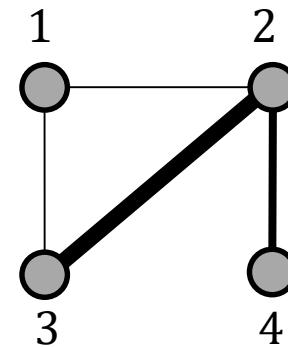
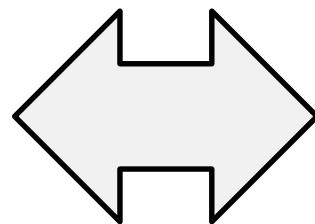
Remark

- Can represent demand matrix as a **demand graph**

sparse distribution \mathcal{D}

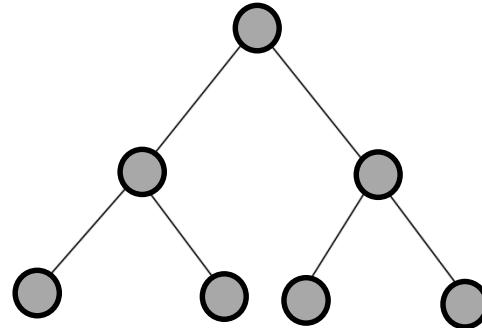


sparse graph $G(\mathcal{D})$



Some Examples

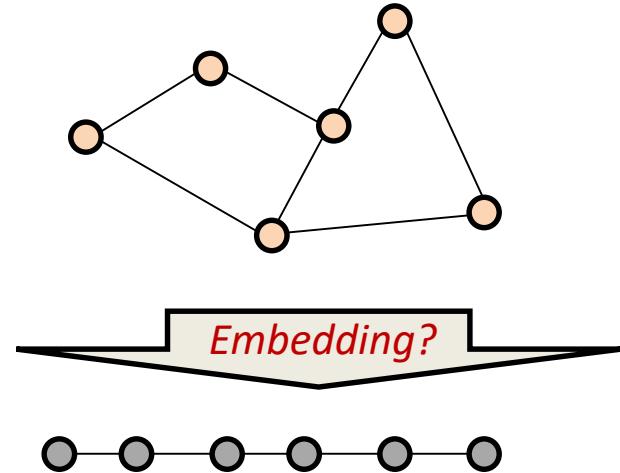
- DANs of $\Delta = 3$:
 - E.g., complete binary **tree**
 - $d_N(u,v) \leq 2 \log n$
 - Can we do **better** than ***log n***?
- DANs of $\Delta = 2$:
 - E.g., set of **lines** and **cycles**



Remark: Hardness Proof

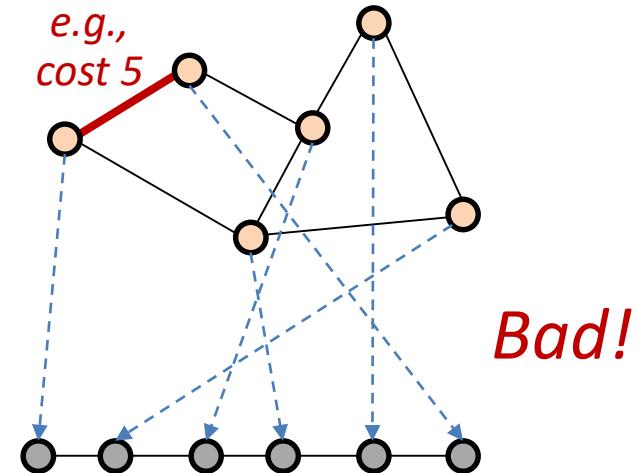
DAN design can be NP-hard

- **Example $\Delta = 2$: A Minimum Linear Arrangement (MLA) problem**
 - A “Virtual Network Embedding Problem”, VNEP
 - *Minimize sum* of lengths of virtual edges



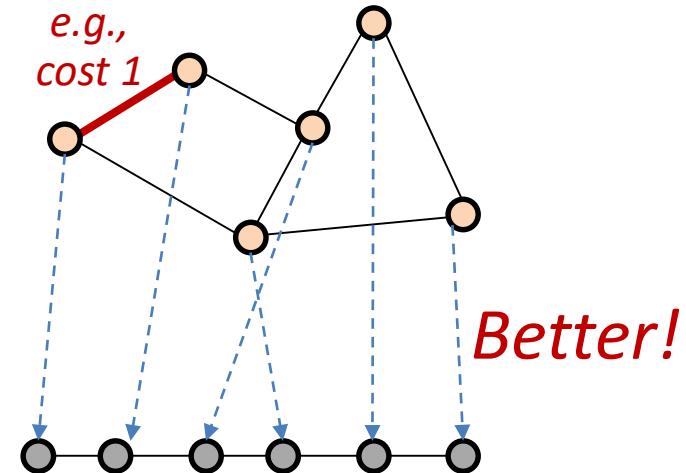
DAN design can be NP-hard

- **Example $\Delta = 2$: A Minimum Linear Arrangement (MLA) problem**
 - A “Virtual Network Embedding Problem”, VNEP
 - *Minimize sum* of lengths of virtual edges



DAN design can be NP-hard

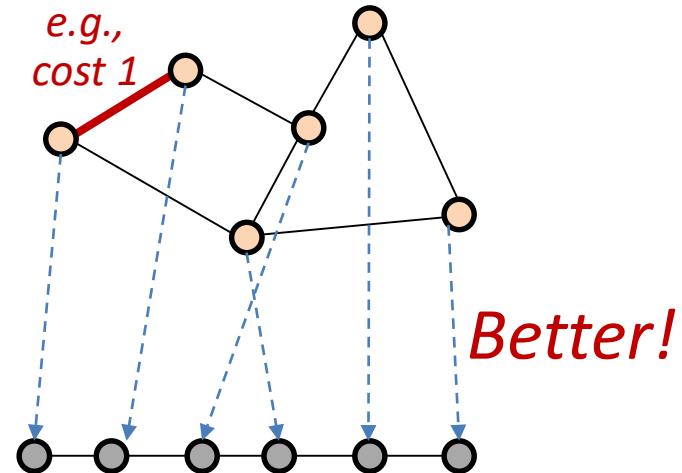
- **Example $\Delta = 2$:** A Minimum Linear Arrangement (**MLA**) problem
 - A “Virtual Network Embedding Problem”, VNEP
 - *Minimize sum* of lengths of virtual edges



DAN design can be NP-hard

- Example $\Delta = 2$: A Minimum Linear Arrangement (MLA) problem
 - A “Virtual Network Embedding Problem”, VNEP
 - *Minimizing the lengths of virtual edges*

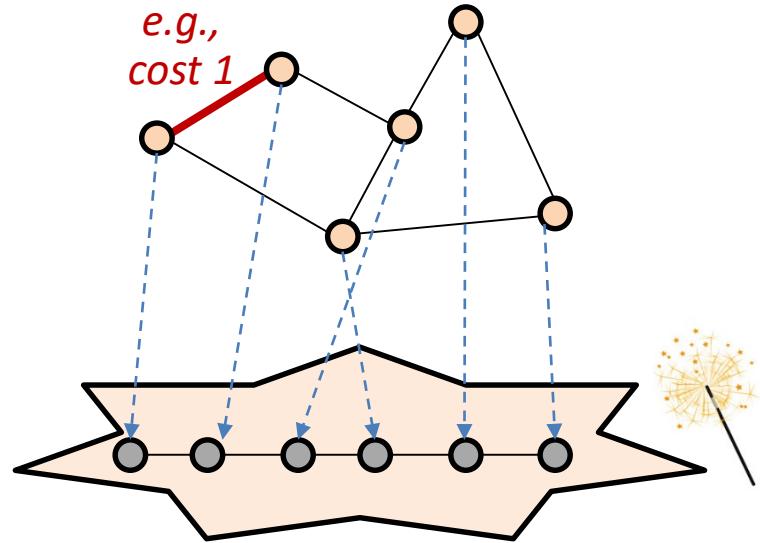
NP-hard, and so is DAN design.



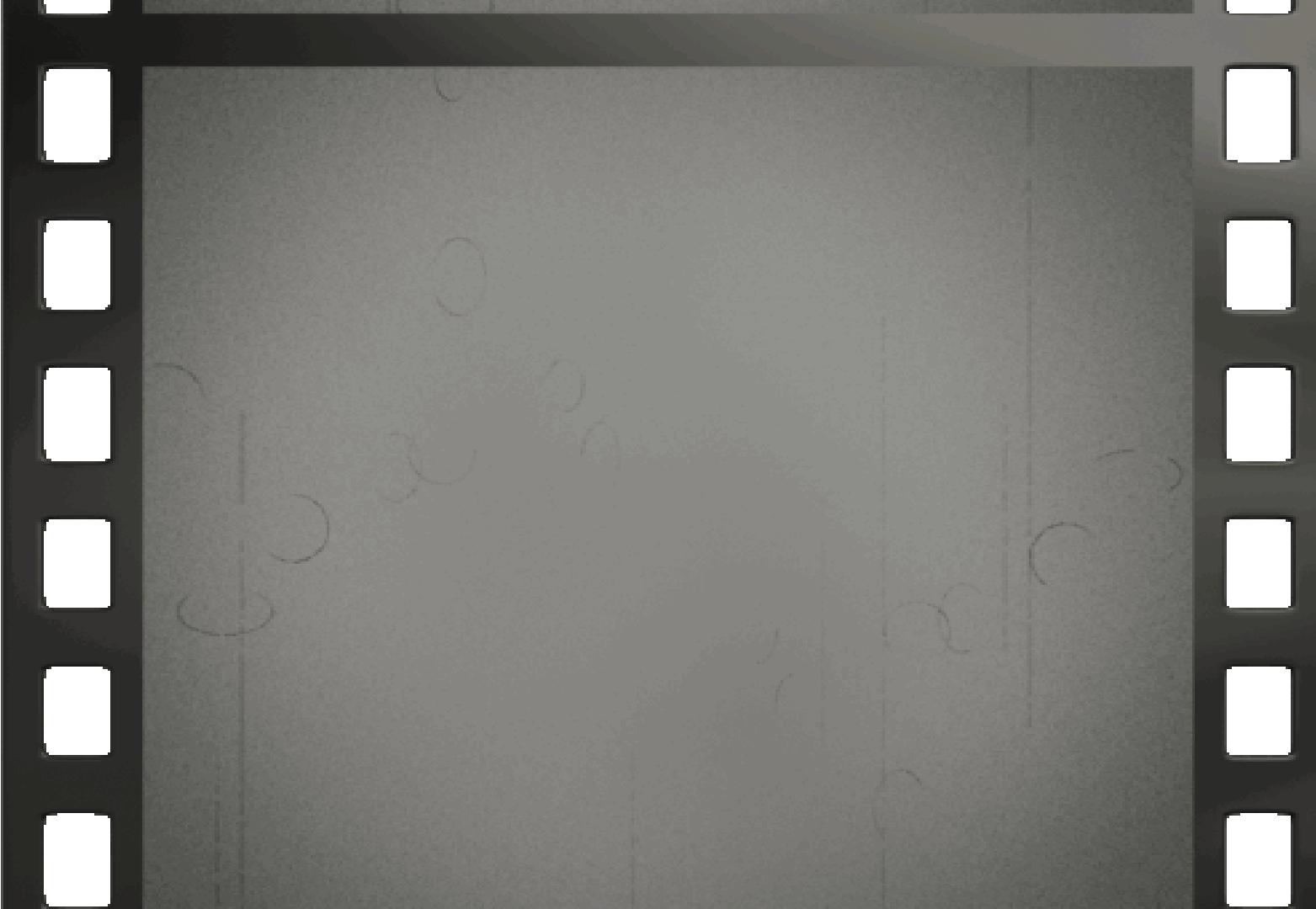
DAN design can be NP-hard

- Example $\Delta = 2$: A Minimum Linear Arrangement (MLA) problem
 - A “Virtual Network Embedding Problem”, VNEP
 - *Minimizing lengths of virtual edges*
- But what about > 2 ? *Embedding* problem still hard, but we have an additional **degree of freedom**:

Do topological flexibilities make problem easier or harder?!



A new knob for optimization!

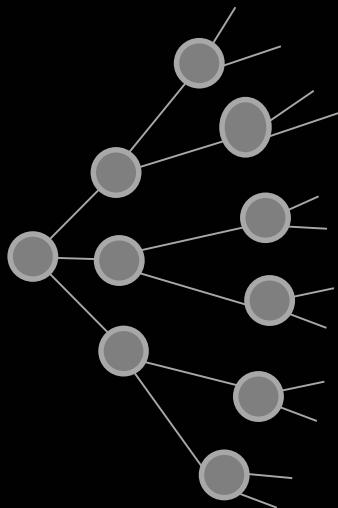


Rewinding the Clock: Degree-Diameter Tradeoff

Each network with n nodes and max degree $\Delta \geq 2$
must have a diameter of at least $\log(n)/\log(\Delta-1) - 1$.

Example: constant Δ , $\log(n)$ diameter

Proof Idea



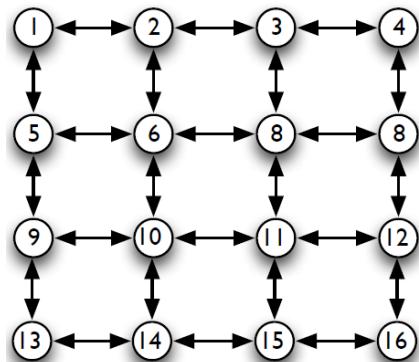
$$1 \quad \Delta \quad \Delta(\Delta - 1) \dots$$

In k steps, reach at
most $1 + \sum \Delta(\Delta - 1)^k$
nodes

Is there a better tradeoff in DANs?

Sometimes, DANs can be much better!

Example 1: low-degree demand

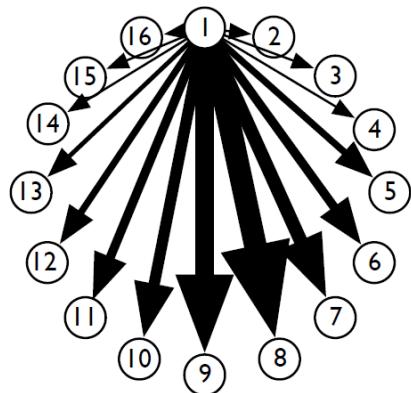


If **demand graph** is of degree Δ , it is trivial to design a **DAN** of degree Δ which achieves an *expected route length of 1*.

Just take DAN =
demand graph!

Sometimes, DANs can be much better!

Example 2: skewed demand

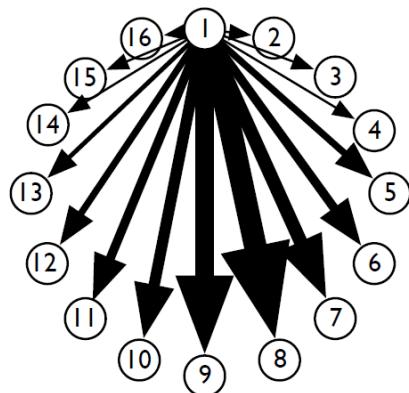


If **demand** is highly skewed, it is also possible to achieve an *expected route length of $O(1)$* in a constant-degree DAN.



Sometimes, DANs can be much better!

Example 2: skewed demand



If **demand** is highly skewed, it is also possible to achieve an ***expected route length of $O(1)$*** in a constant-degree DAN.



E.g., arrange neighbors of node 1 in a **Huffman tree**!

So on what does it depend?

So on what does it depend?



We argue (but still don't know!): on the
“entropy” of the demand!



SPEED
LIMIT
?

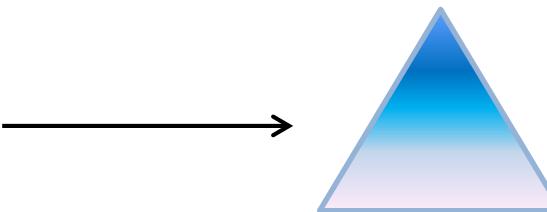
Intuition: Entropy Lower Bound



Lower Bound Idea: Leverage Coding or Datastructure

		Destinations						
		1	2	3	4	5	6	7
Sources	1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
	2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
	3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
	4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
	5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
	6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
	7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

- DAN just for a *single (source) node 3*



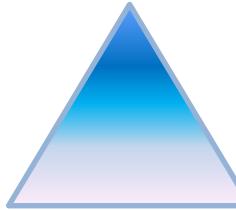
- How good can this tree be? Cannot do better than Δ -ary **Huffman tree** for its destinations
- Entropy** lower bound on ERL known for binary trees, e.g. **Mehlhorn** 1975

Lower Bound Idea: Leverage Coding or Datastructure

		Destinations						
		1	2	3	4	5	6	7
Sources	1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$
	2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$
	3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$
	4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0
	5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0
	6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$
	7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0

An optimal “ego-tree”
for this source!

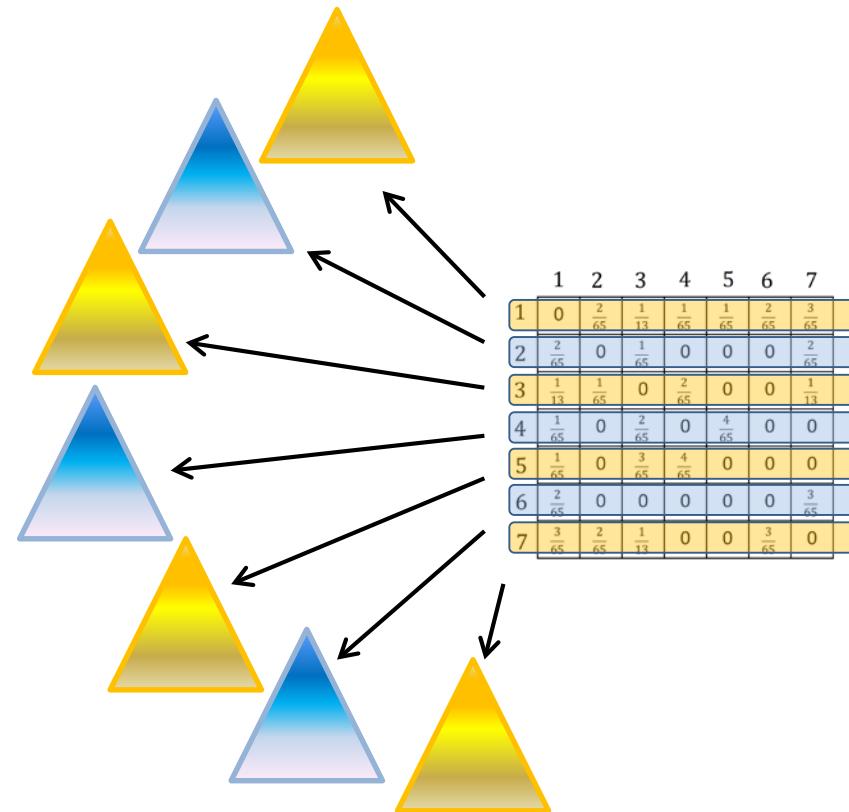
- DAN just for a *single (source) node 3*



- How good can this tree be? Cannot do better than Δ -ary **Huffman tree** for its destinations
- Entropy** lower bound on ERL known for binary trees, e.g. **Mehlhorn** 1975

So: Entropy of the *Entire* Demand

- Proof idea ($EPL = \Omega(H_{\Delta}(Y|X))$):
 - sources
 - destinations
 - entropy
 - degree
- Compute **ego-tree** for each source node
- Take **union** of all **ego-trees**
- Violates **degree restriction** but valid lower bound



Entropy of the *Entire* Demand: Sources and Destinations

Do this in **both dimensions**:

$$\text{EPL} \geq \Omega(\max\{\mathcal{H}_\Delta(Y|X), \mathcal{H}_\Delta(X|Y)\})$$

$\Omega(\mathcal{H}_\Delta(X Y))$							$\Omega(\mathcal{H}_\Delta(Y X))$	
		1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$	
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$	
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$	
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0	
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0	
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$	
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0	

\mathcal{D}

Entropy of the *Entire* Demand: Sources and Destinations

Do this in **both dimensions**:

$$\text{EPL} \geq \Omega(\max\{\mathcal{H}_\Delta(Y|X), \mathcal{H}_\Delta(X|Y)\})$$

$\Omega(\mathcal{H}_\Delta(X Y))$							$\Omega(\mathcal{H}_\Delta(Y X))$	
		1	2	3	4	5	6	7
1	0	$\frac{2}{65}$	$\frac{1}{13}$	$\frac{1}{65}$	$\frac{1}{65}$	$\frac{2}{65}$	$\frac{3}{65}$	
2	$\frac{2}{65}$	0	$\frac{1}{65}$	0	0	0	$\frac{2}{65}$	
3	$\frac{1}{13}$	$\frac{1}{65}$	0	$\frac{2}{65}$	0	0	$\frac{1}{13}$	
4	$\frac{1}{65}$	0	$\frac{2}{65}$	0	$\frac{4}{65}$	0	0	
5	$\frac{1}{65}$	0	$\frac{3}{65}$	$\frac{4}{65}$	0	0	0	
6	$\frac{2}{65}$	0	0	0	0	0	$\frac{3}{65}$	
7	$\frac{3}{65}$	$\frac{2}{65}$	$\frac{1}{13}$	0	0	$\frac{3}{65}$	0	

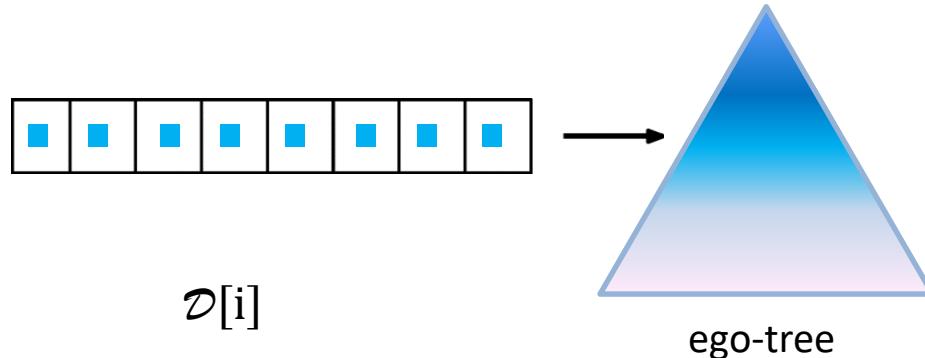
\mathcal{D}

Achieving Entropy Limit: Algorithms



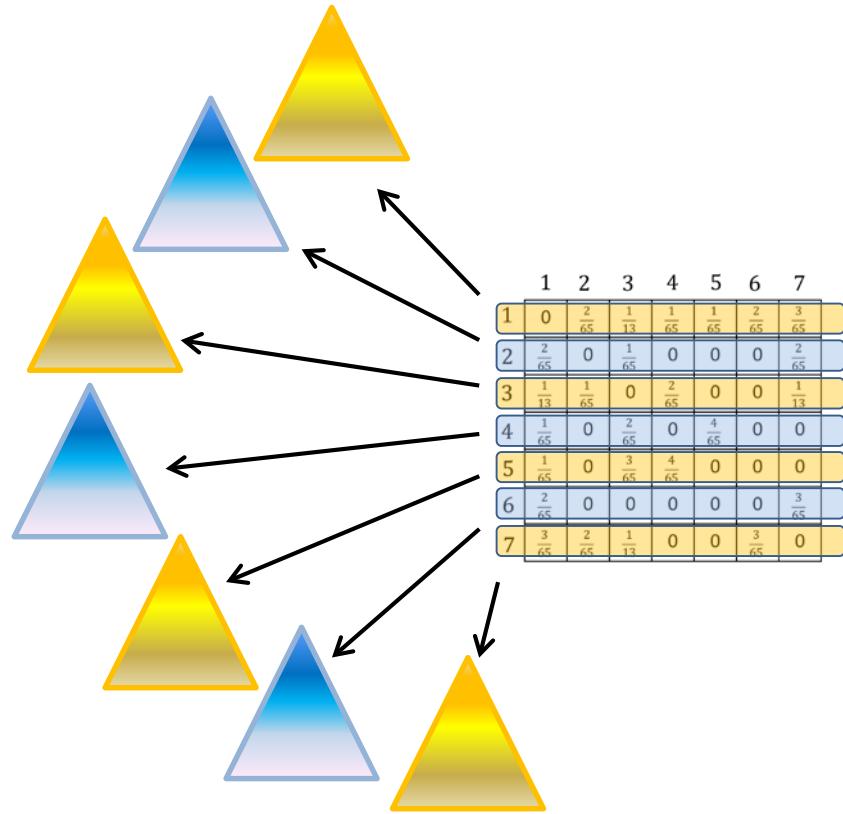
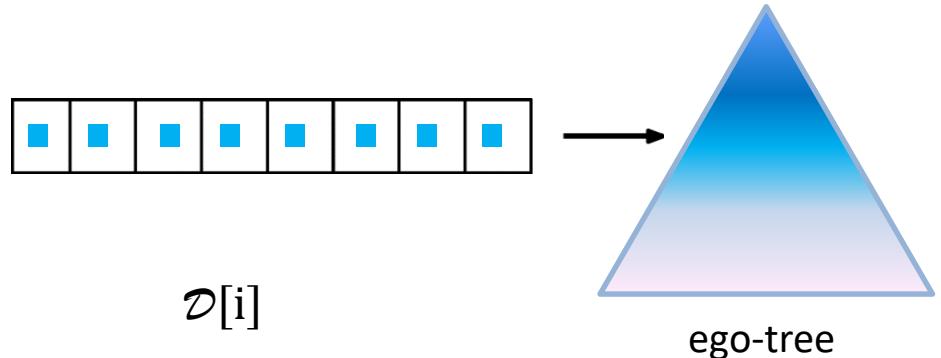
Ego-Trees Revisited

- ego-tree: optimal tree for a row (= given source)



Ego-Trees Revisited

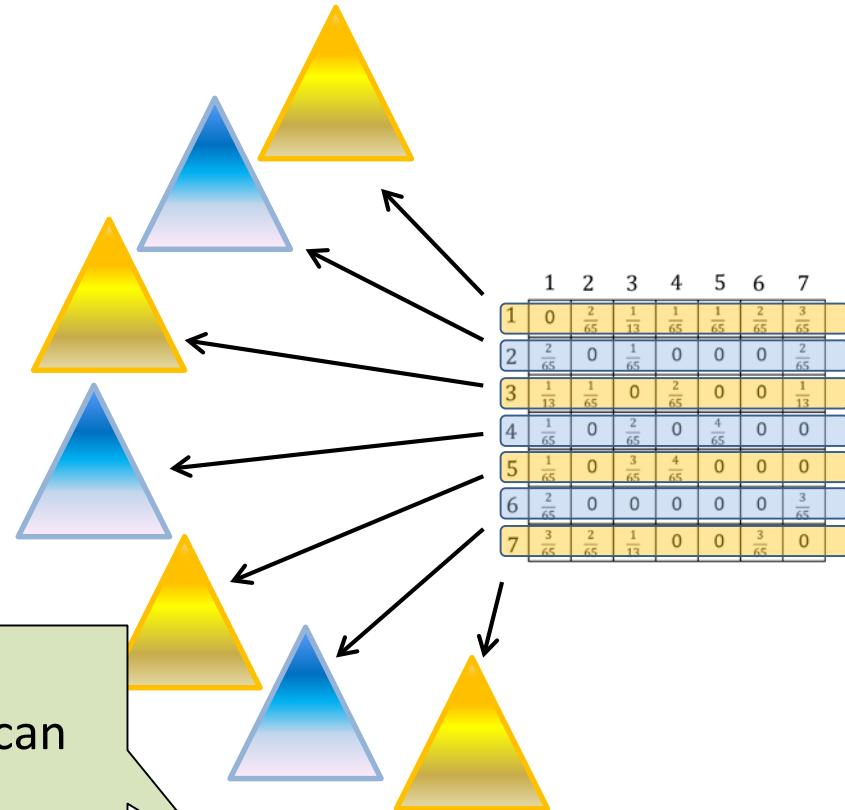
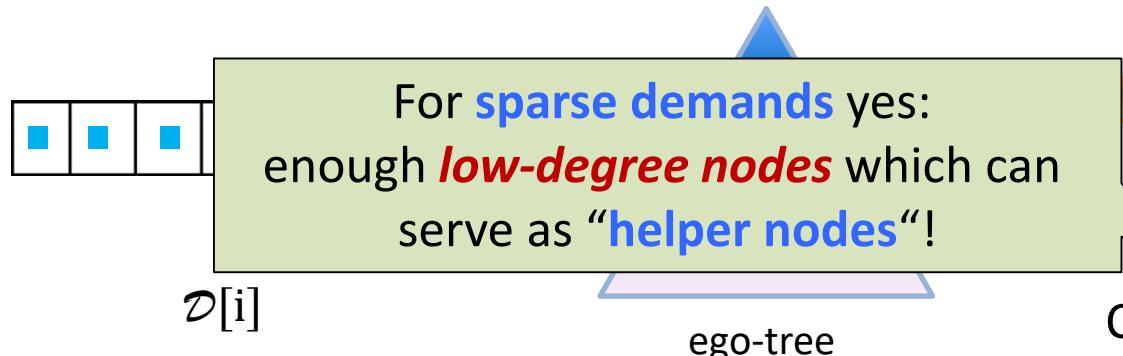
- ego-tree: optimal tree for a row (= given source)



Can we merge the trees **without distortion** and **keep degree low**?

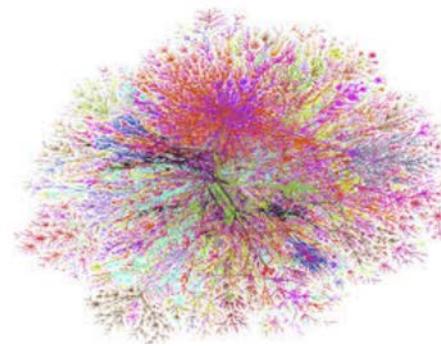
Ego-Trees Revisited

- ego-tree: optimal tree for a row (= given source)



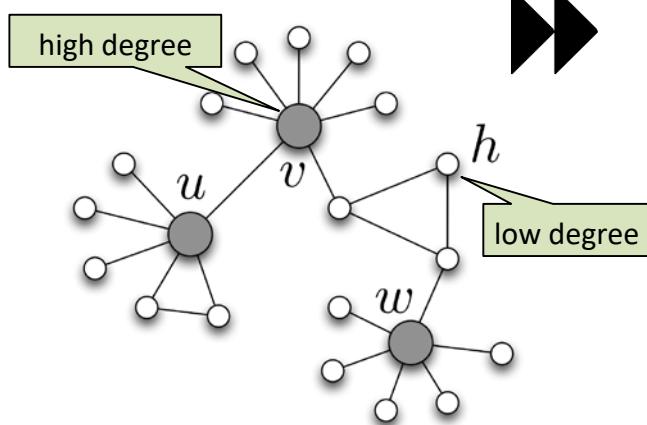
Can we merge the trees **without distortion** and **keep degree low**?

From Trees to Networks

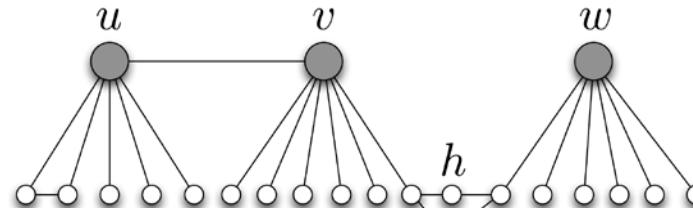


Idea: Degree Reduction

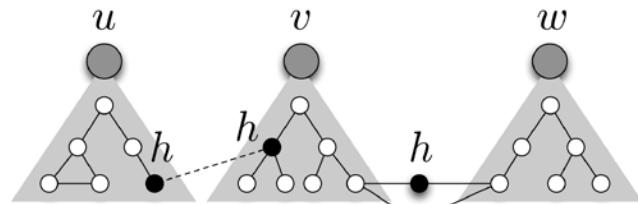
① Demand graph



② Hierarchical representation



③ Add low-degree nodes as helpers



Taking union of ego-trees results in **high degree**:
 u and v will appear in many ego-trees

Node h **helps edge (u, v)** by participating in *ego-tree(u)* as a relay node toward v and *in ego-tree(v)* as a relay toward u



DAN Design: Related to Spanners

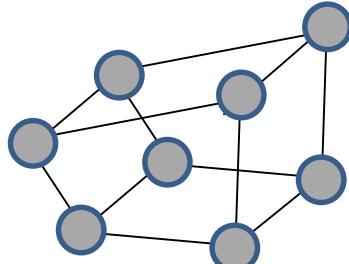
Low-Distortion Spanners

- Classic problem: find sparse, distance-preserving (low-distortion) **spanner** of a graph
- But:
 - Spanners aim at low distortion among ***all pairs***; in our case, we are only interested in the ***local distortion***, 1-hop communication neighbors
 - We allow **auxiliary edges** (not a subgraph): similar to **geometric spanners**
 - We require ***constant degree***

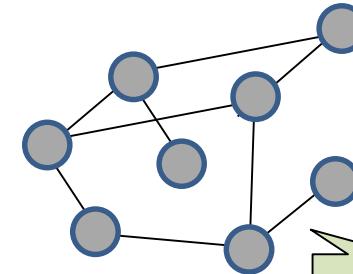
Yet: We can leverage the connection to spanners sometimes!

Theorem: If request distribution \mathcal{D} is **regular and uniform**, and if we can find a constant distortion, linear sized (i.e., **constant, sparse**) spanner for this request graph: then we can design a constant degree DAN providing an **optimal ERL** (i.e., $O(H(X|Y)+H(Y|X))$).

r-regular and uniform
demand:

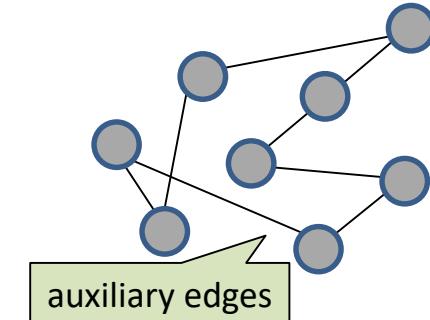


***Sparse, irregular
(constant) spanner:***



subgraph!

***Constant degree optimal
DAN (ERL at most log r):***



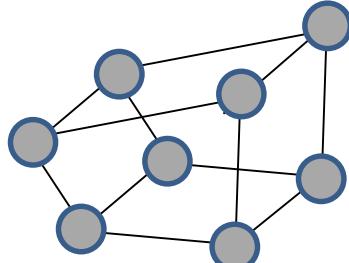
auxiliary edges

Yet: We can leverage the connection to spanners sometimes!

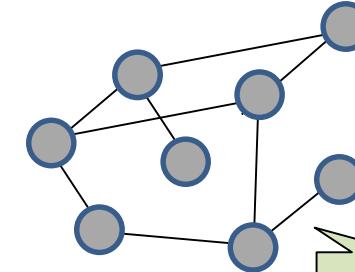
Theorem: If request distribution \mathcal{D} is **regular and uniform**, and if we can find a constant distortion, linear sized (i.e., **constant sparse**) spanner for this request graph: then we can design a constant degree DAN

Simply using our degree reduction trick again: now for spanner!

r-regular and uniform
demand:

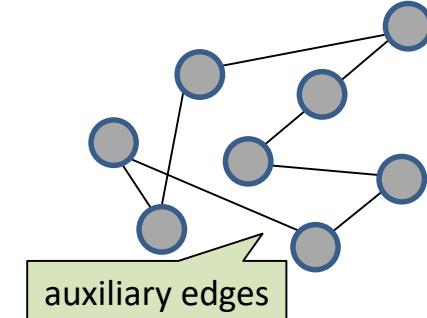


***Sparse, irregular
(constant) spanner:***



subgraph!

Constant degree optimal
DAN (ERL at most ***log r***):

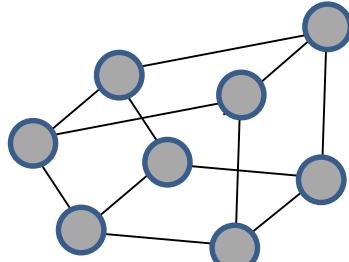


auxiliary edges

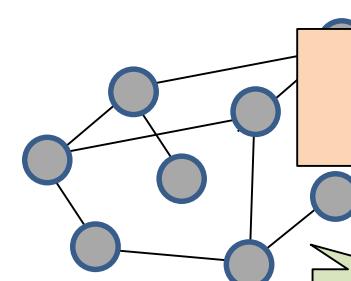
Yet: We can leverage the connection to spanners sometimes!

Theorem: If request distribution \mathcal{D} is **regular and uniform**, and if we can find a constant distortion, linear sized (i.e., **constant, sparse**) spanner for this request graph: then we can design a constant degree DAN providing an **optimal EPL** (i.e., $O(H(X|Y) + H(Y|X))$).

r-regular and uniform
demand:



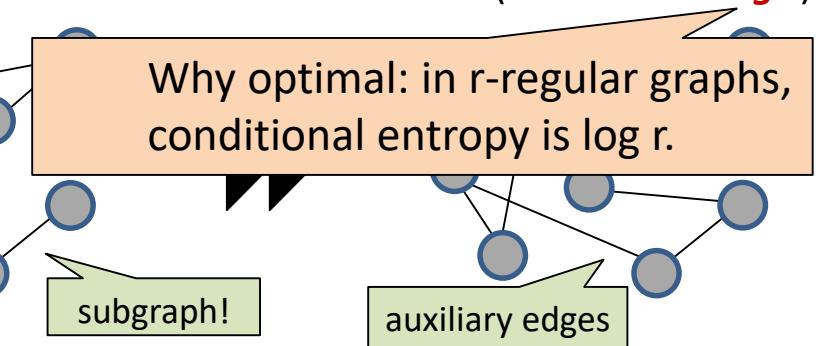
*Sparse, irregular
(constant) spanner:*



Why optimal: in r -regular graphs,
conditional entropy is $\log r$.

subgraph!

Constant degree optimal
DAN (ERL at most **$\log r$**):

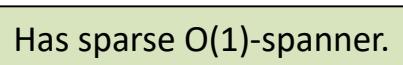


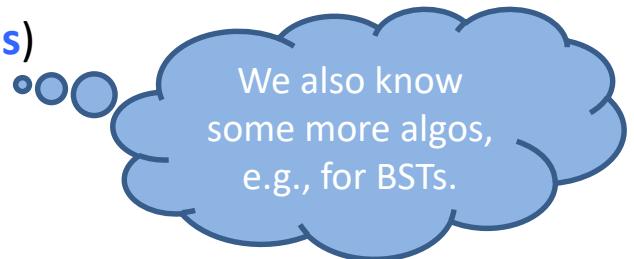
auxiliary edges

Proof Idea

- **Degree reduction** again, this time *from sparse spanner* (before: from sparse demand graph)

Corollaries

- Optimal DAN designs for
 - **Hypercubes** (with $n \log n$ edges)
 - **Chordal graphs**  Has sparse $O(1)$ -spanner.
 - Trivial: graphs with polynomial degree (**dense graphs**)
 - Graphs of **locally bounded doubling dimension**



So: How *much* structure/entropy is there?



How to *measure* it?
And which *types of structures*? E.g., *temporal*
structure in addition to *non-temporal* structure?
More *tricky*!

Often only intuitions in the literature...

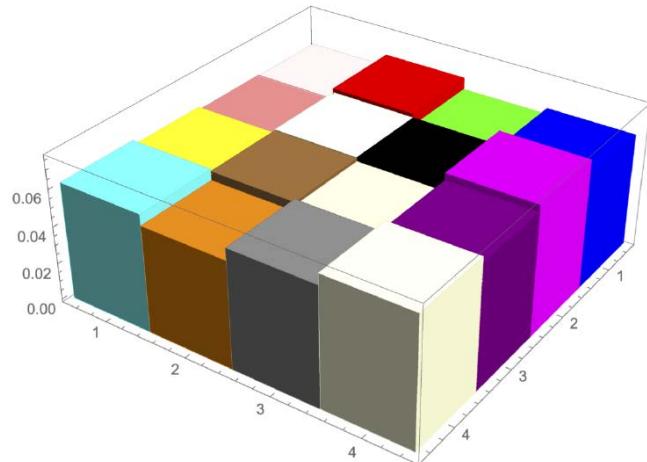
“less than 1% of the rack pairs account for 80% of the total traffic”

“only a few ToRs switches are hot and most of their traffic goes to a few other ToRs”

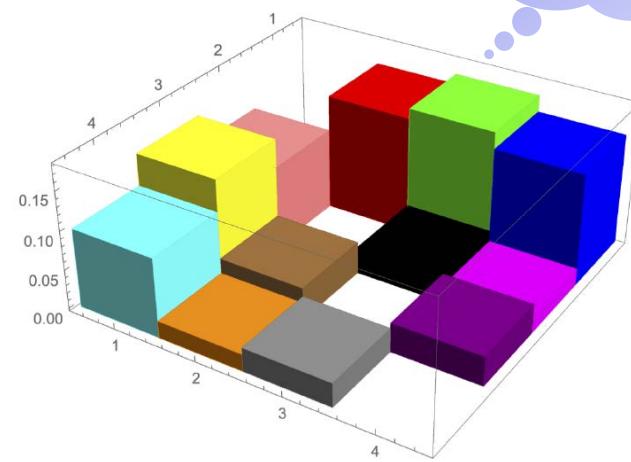
“over 90% bytes flow in elephant flows”

... and it *is* intuitive!

Non-temporal Structure



VS



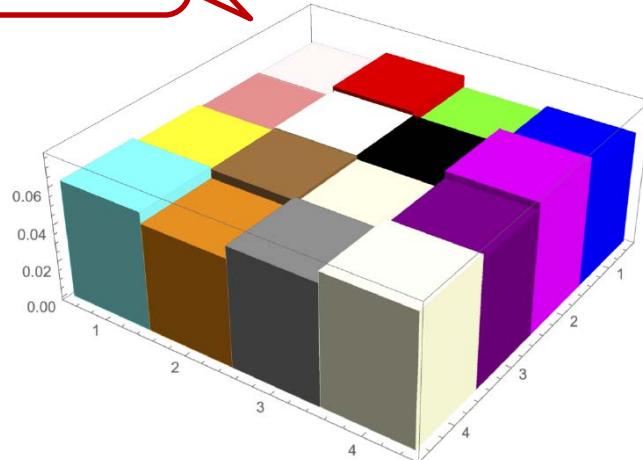
Color =
comm. pair

Traffic matrix of two different **distributed ML** applications (GPU-to-GPU):
Which one has *more structure*?

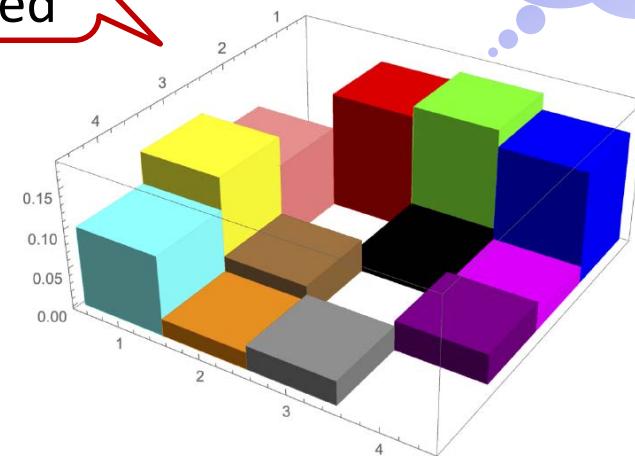
... and it *is* intuitive!

Non-temporal Structure

More uniform



More skewed



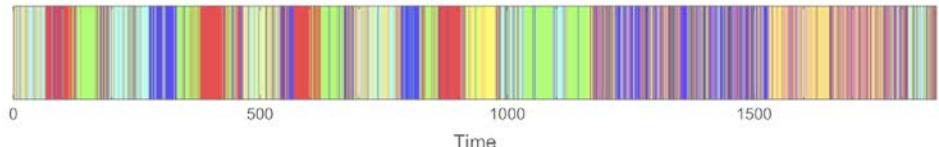
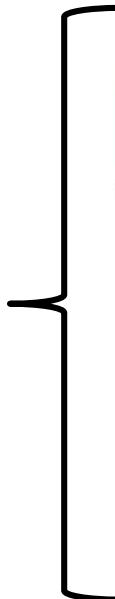
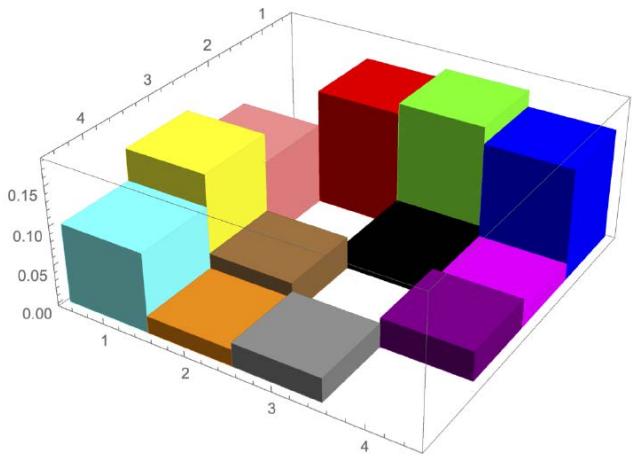
VS

Color =
comm. pair

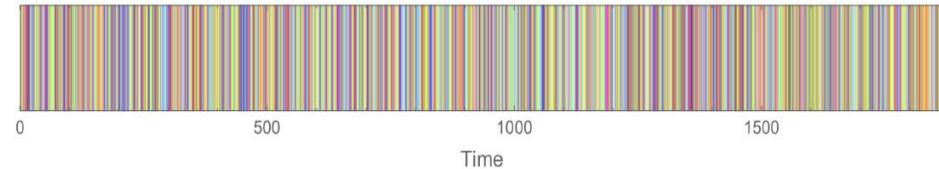
Traffic matrix of two different **distributed ML** applications (GPU-to-GPU):
Which one has *more structure*?

... and it *is* intuitive!

Temporal Structure



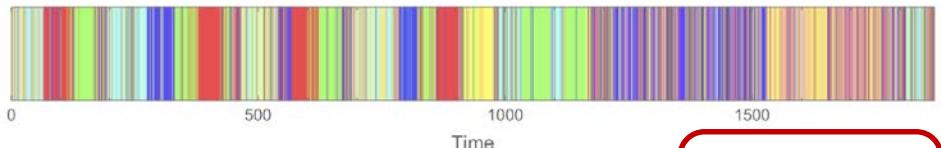
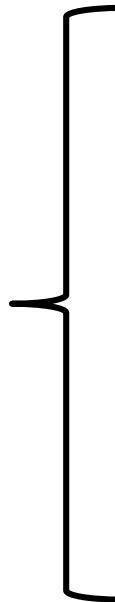
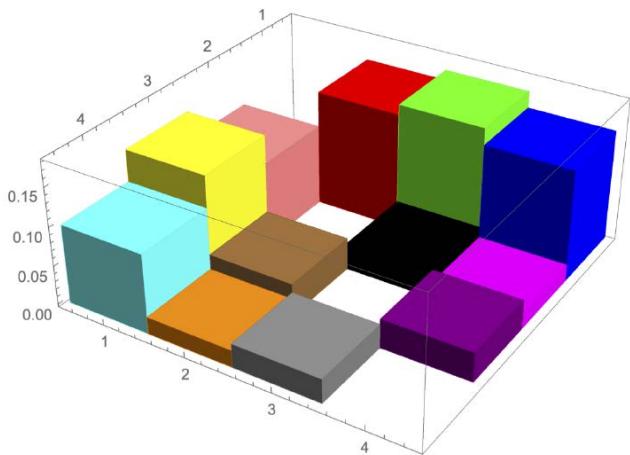
VS



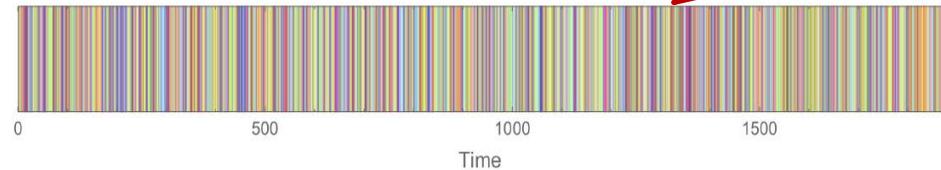
Two different ways to generate *same traffic matrix* (same non-temporal structure):
Which one has *more structure*?

... and it *is* intuitive!

Temporal Structure



VS



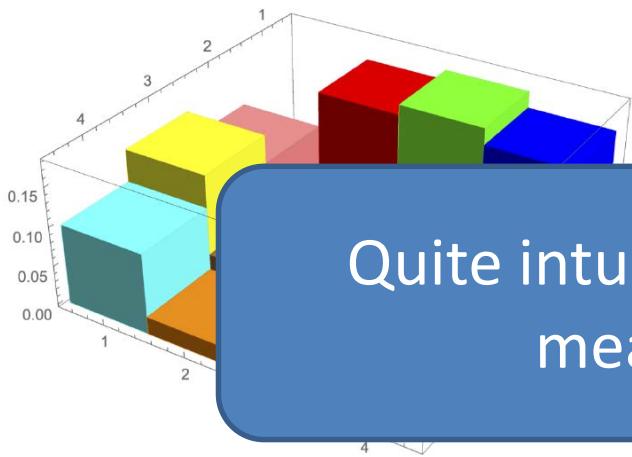
More
bursty

More
random

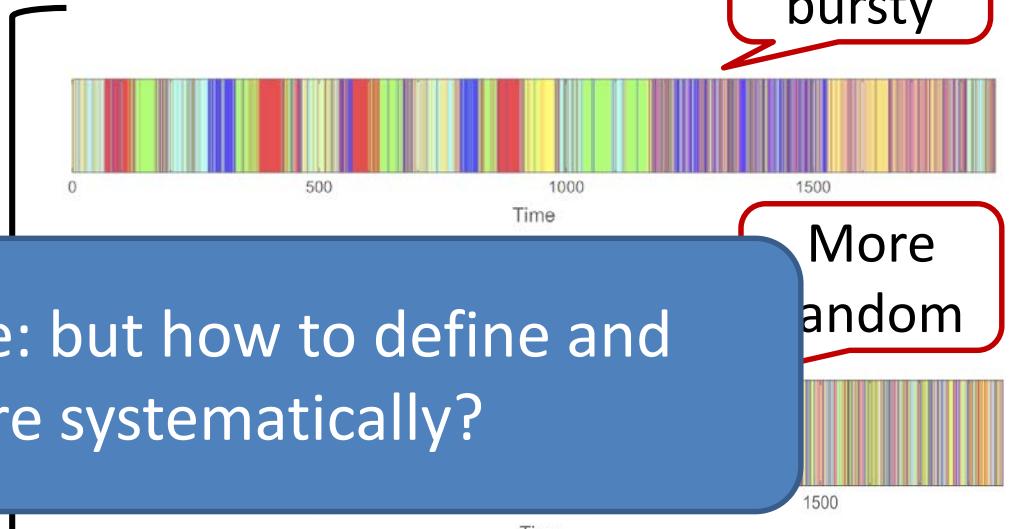
Two different ways to generate *same traffic matrix* (same non-temporal structure):
Which one has *more structure*?

... and it *is* intuitive!

Temporal Structure



Quite intuitive: but how to define and measure systematically?



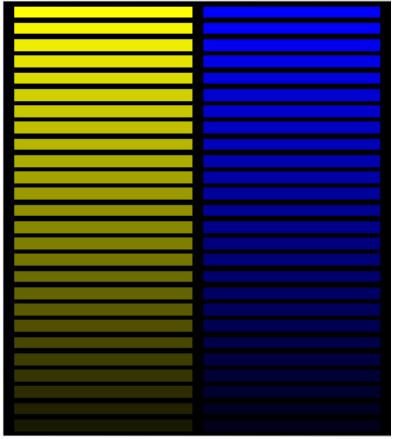
Two different ways to generate *same traffic matrix* (same non-temporal structure):
Which one has *more structure*?

The Trace Complexity

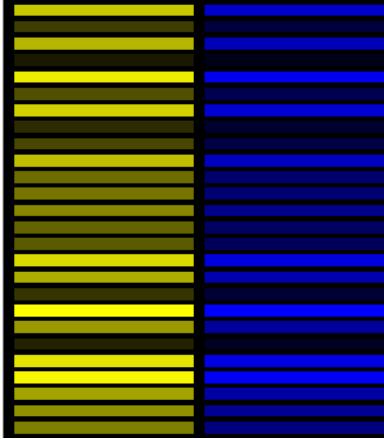
- An **information-theoretic** approach: how can we ***measure the entropy*** (rate) of a traffic trace?
- Henceforth called the **trace complexity**
- Simple approximation: „**shuffle&compress**“
 - Remove structure by iterative ***randomization***
 - Difference of compression ***before and after*** randomization: structure

The Trace Complexity

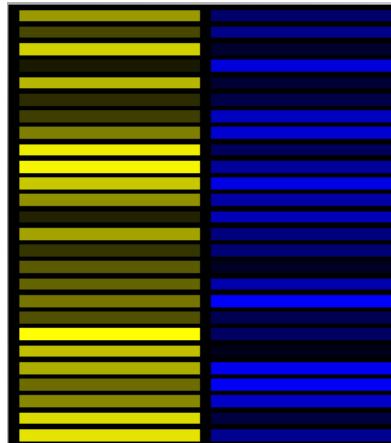
Original src-dst trace



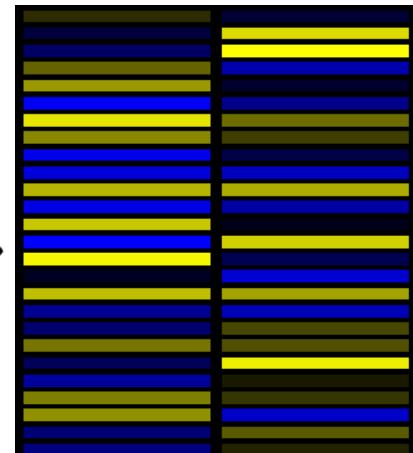
Randomize rows



Randomized columns



Uniform trace

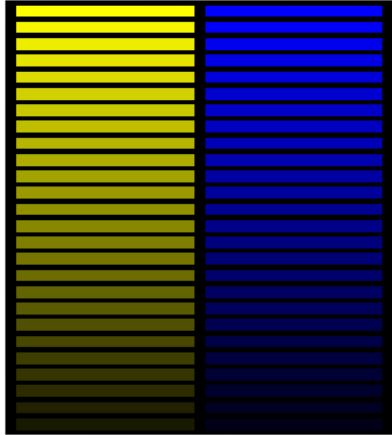


Increasing complexity (systematically randomized)

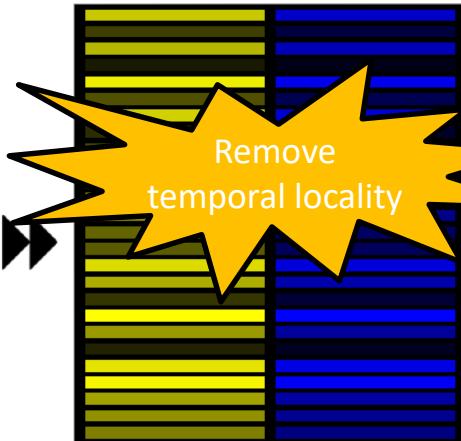
More structure (compresses better)

The Trace Complexity

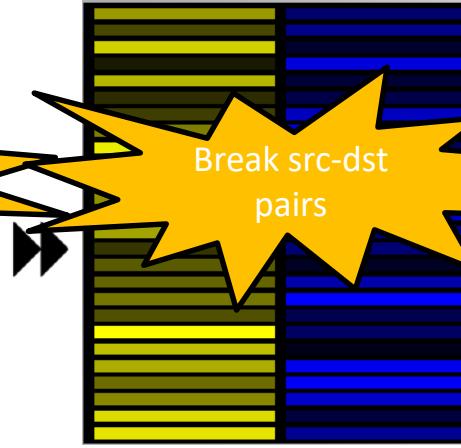
Original src-dst trace



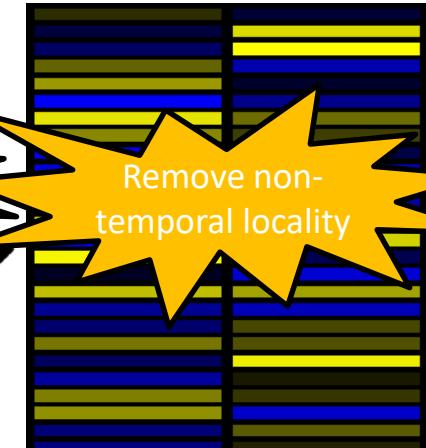
Randomize rows



Randomized columns



Uniform trace



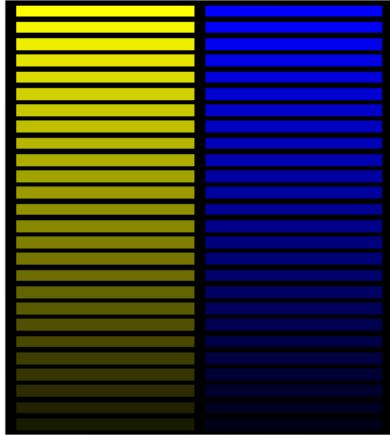
Difference in
compression?

Difference in
compression?

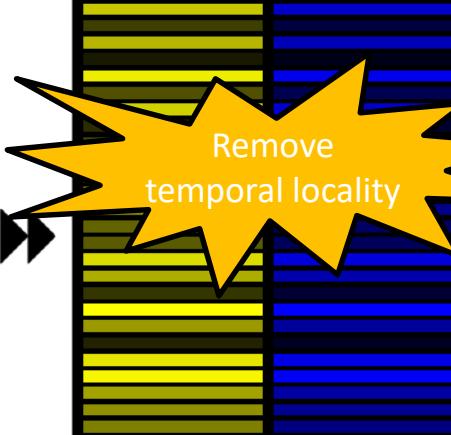
Difference in
compression?

The Trace Complexity

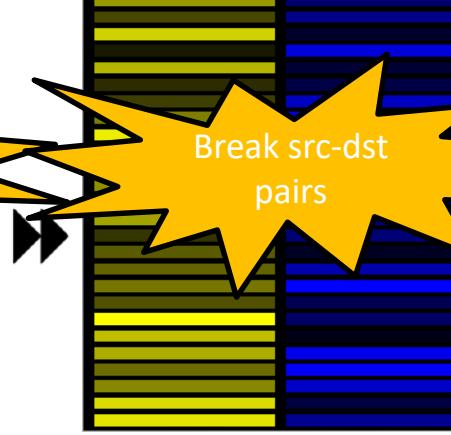
Original src-dst trace



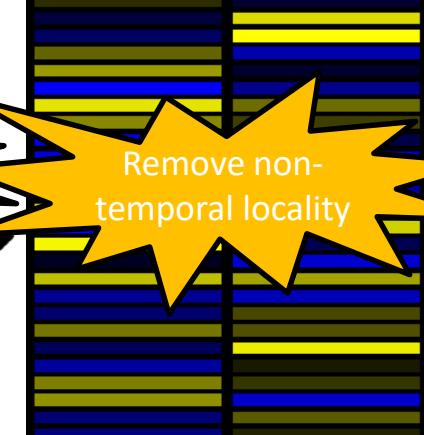
Randomize rows



Randomized columns



Uniform trace



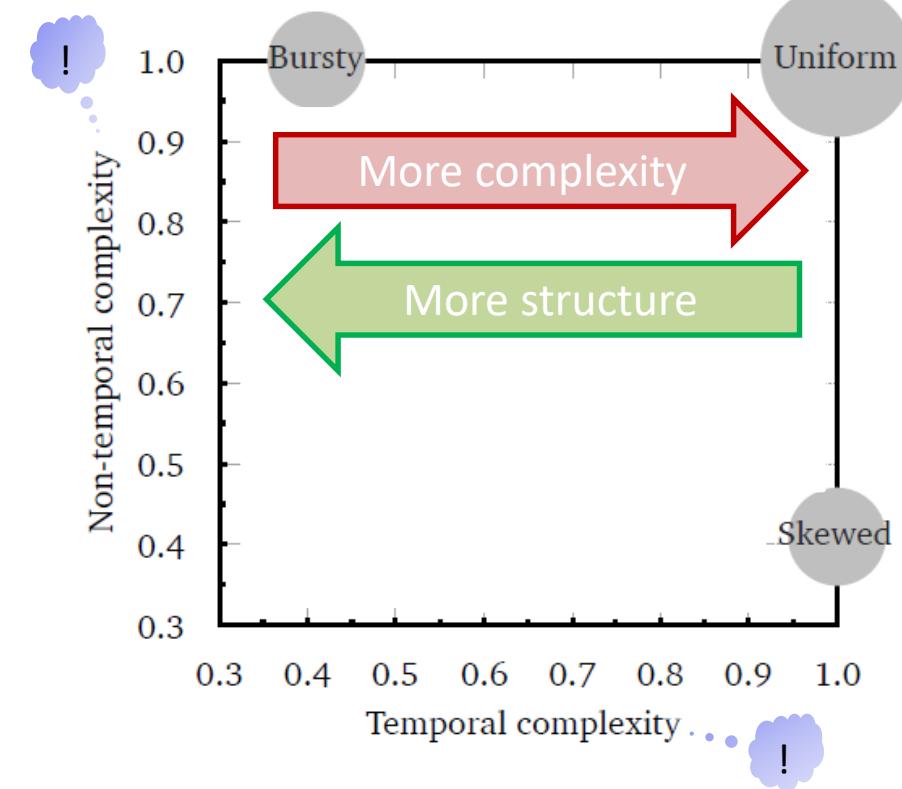
Difference in
compression?

Difference in
compression?

Difference in
compression?

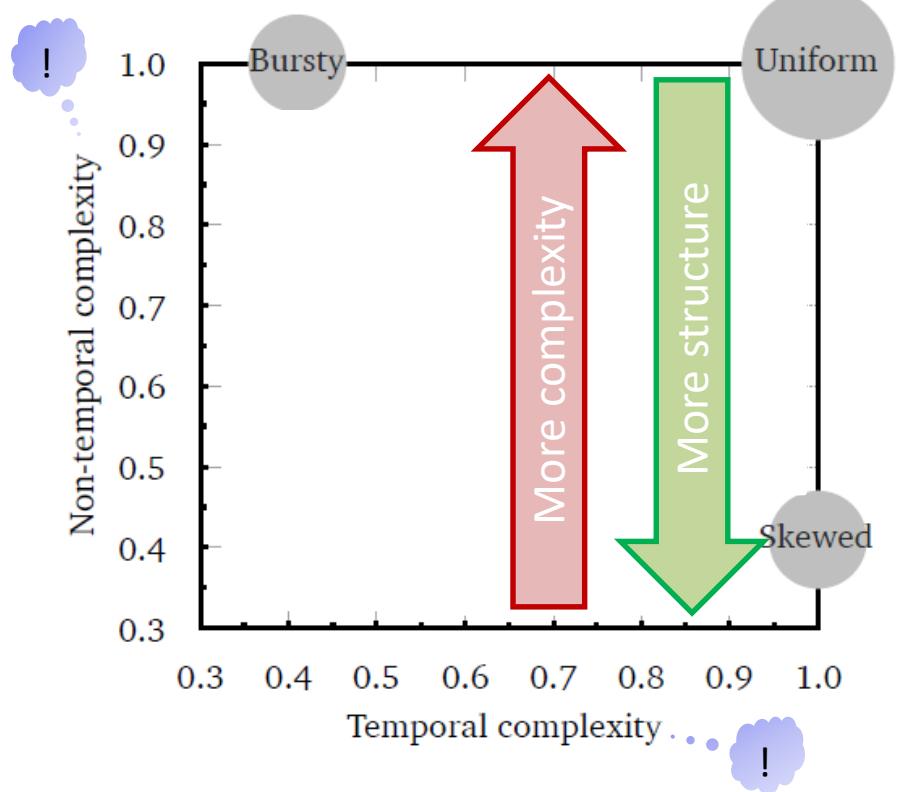
Can be used to define a „complexity map“!

The Complexity Map



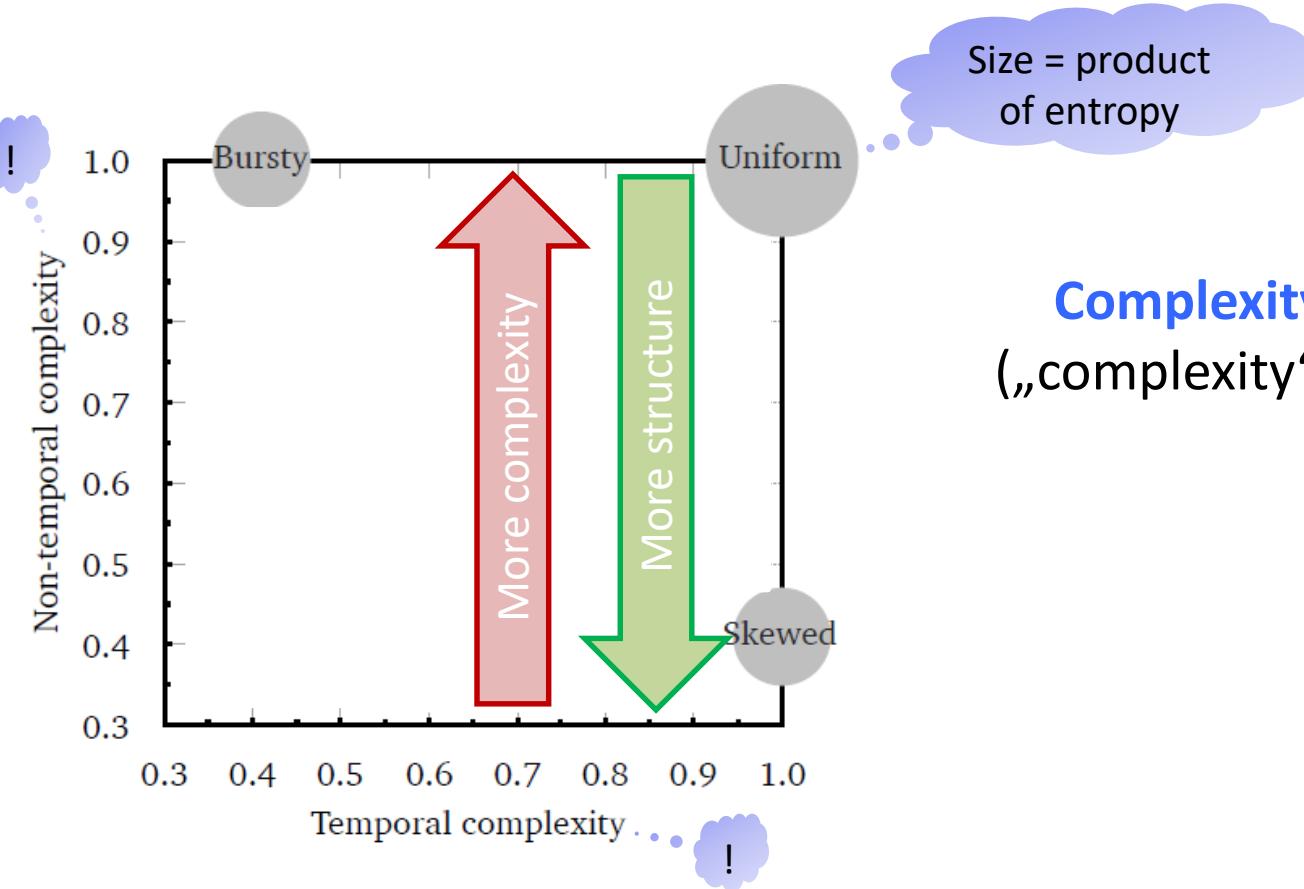
Complexity Map: Entropy („complexity“) of traffic traces.

The Complexity Map

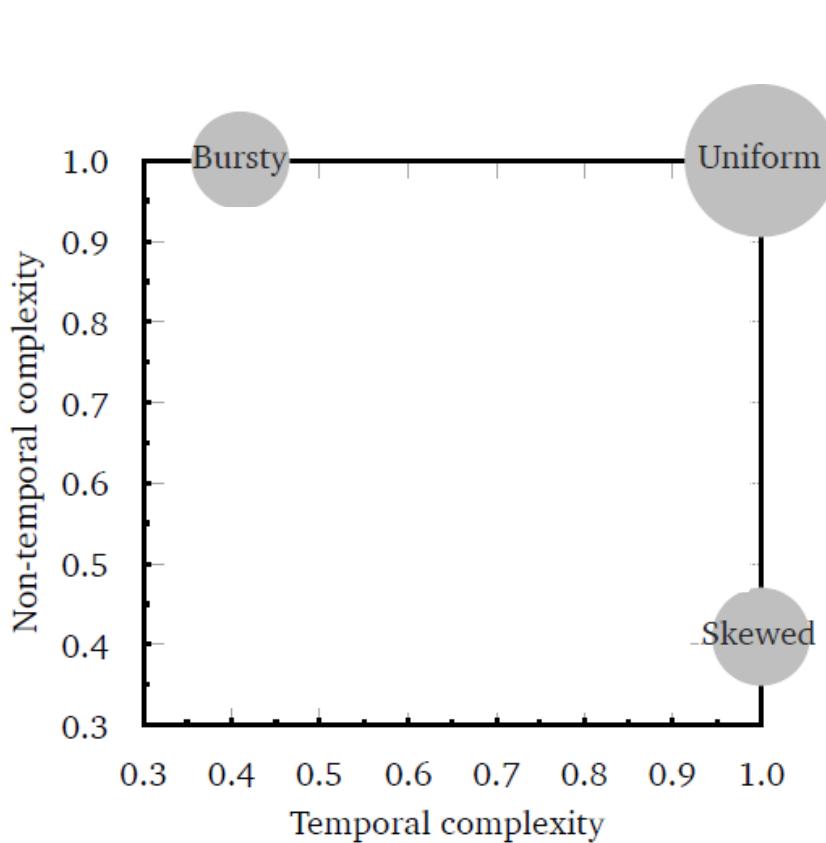


Complexity Map: Entropy
("complexity") of traffic traces.

The Complexity Map

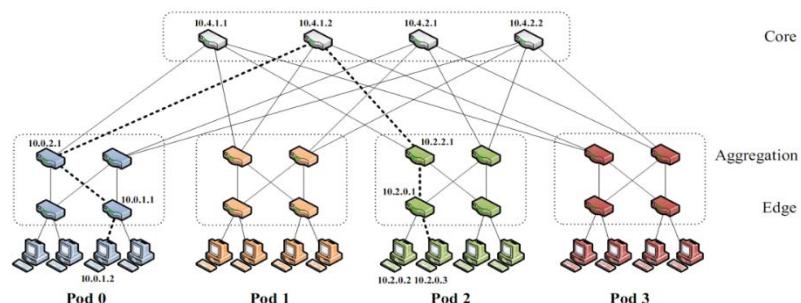


The Complexity Map

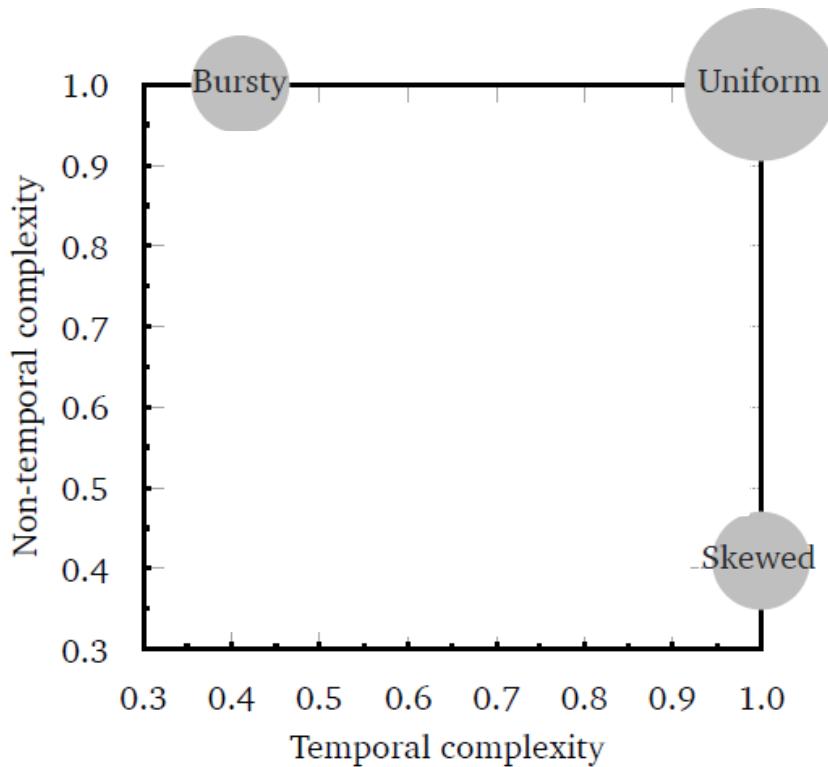


Uniform: Today's datacenters

- Traditional networks are optimized **for the “worst-case” (all-to-all communication traffic)**
- Example, fat-tree topologies: provide **full bisection bandwidth**

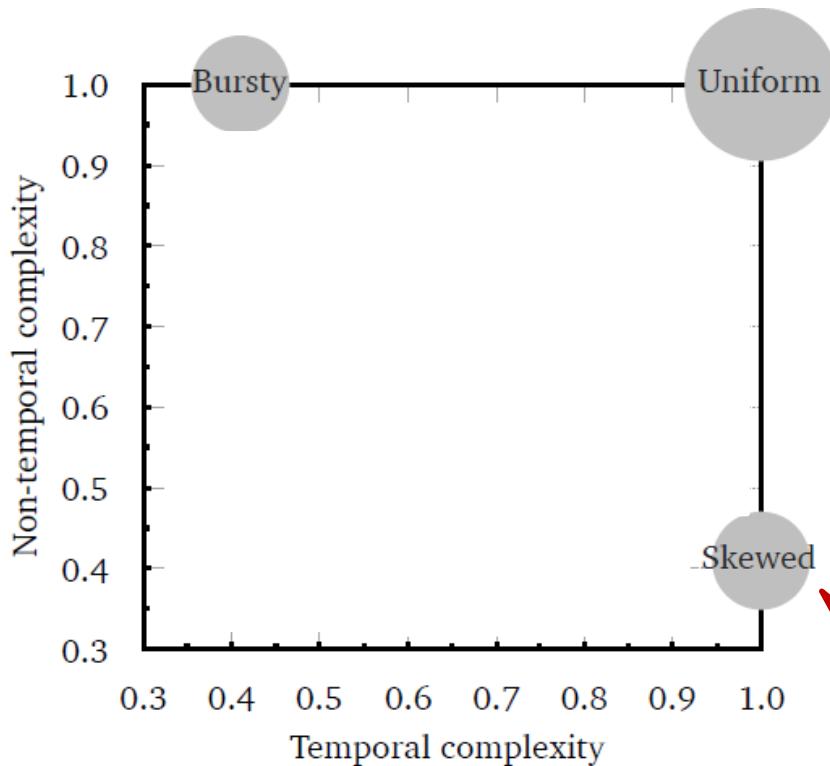


The Complexity Map



Good in the worst case ***but***:
cannot leverage different
temporal and **non-temporal**
structures of traffic traces!

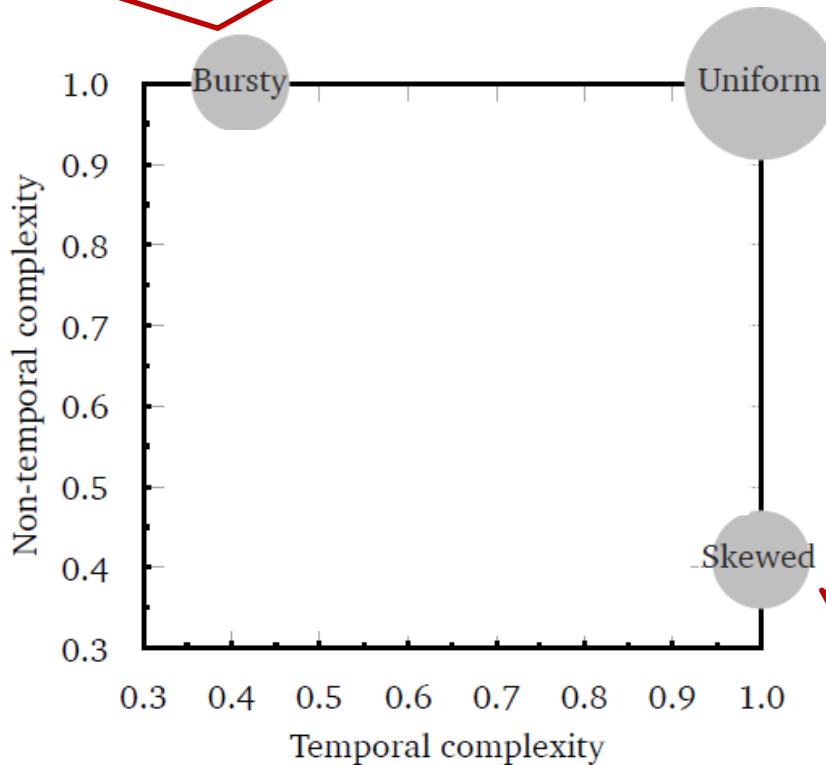
The Complexity Map



Good in the worst case ***but:***
cannot leverage different
temporal and **non-temporal**
structures of traffic traces!

Non-temporal structure could
be exploited already with ***static***
demand-aware networks!

To exploit **temporal** structure,
need ***adaptive demand-aware***
("self-adjusting") networks.

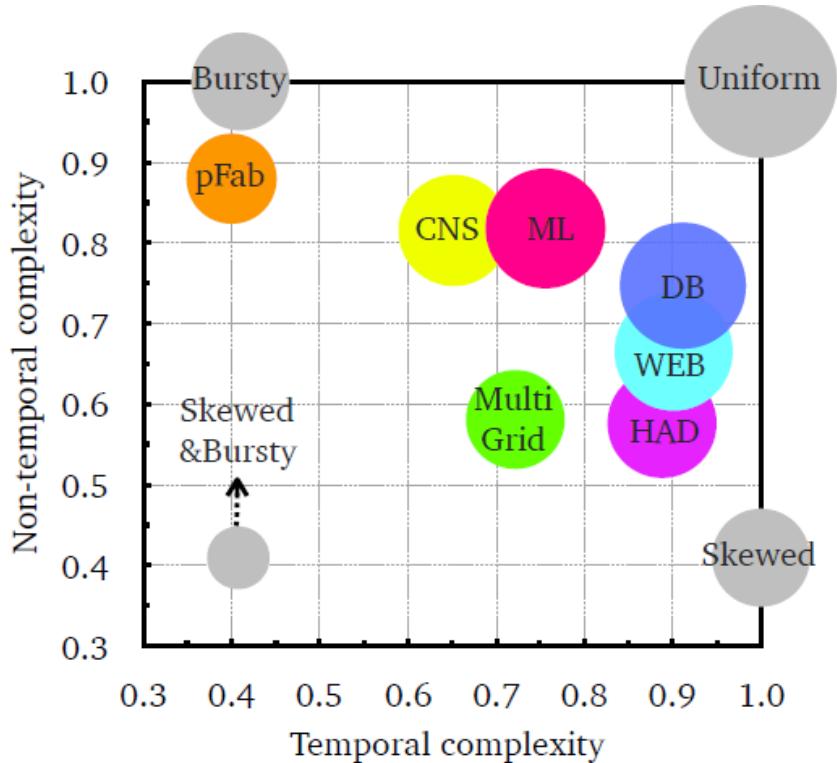


Complexity Map

Good in the worst case ***but:***
cannot leverage different
temporal and **non-temporal**
structures of traffic traces!

Non-temporal structure could
be exploited already with ***static***
demand-aware networks!

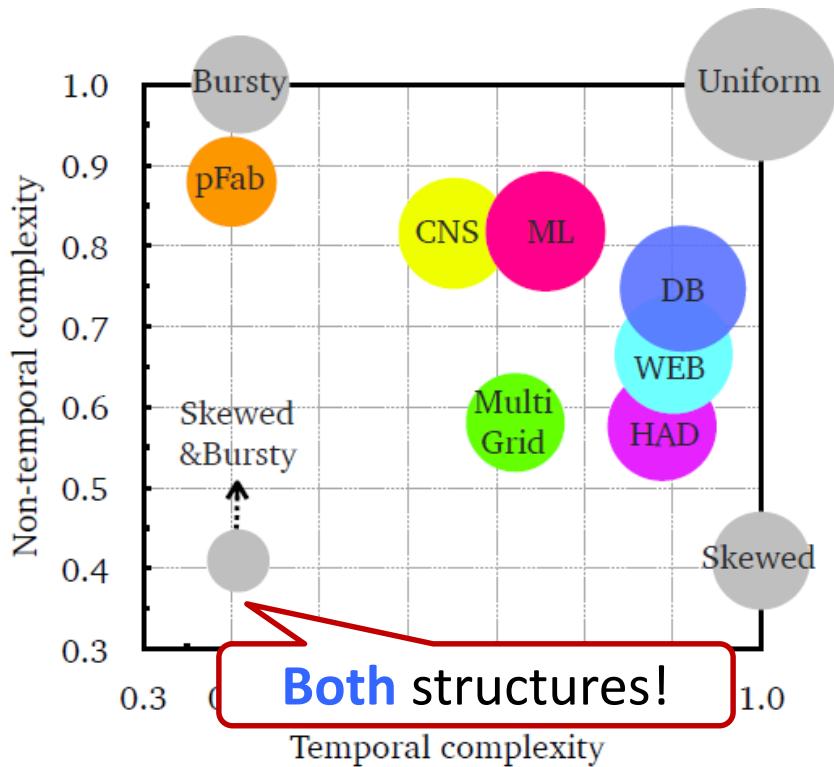
The Complexity Map



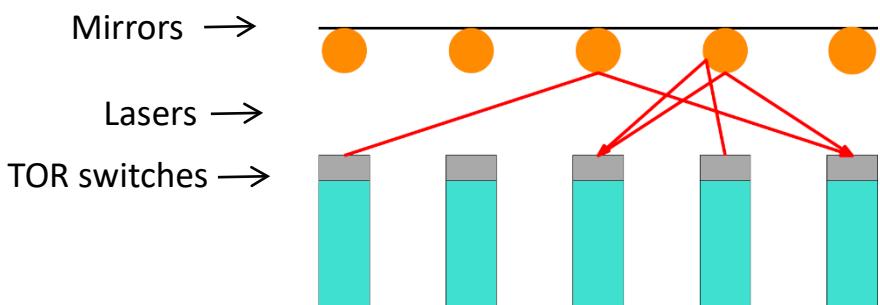
Observation: different applications feature quite significant (and different!) **temporal** and **non-temporal** structures.

- Facebook clusters: DB, WEB, HAD
- HPC workloads: CNS, Multigrid
- Distributed Machine Learning (ML)
- Synthetic traces like pFabric

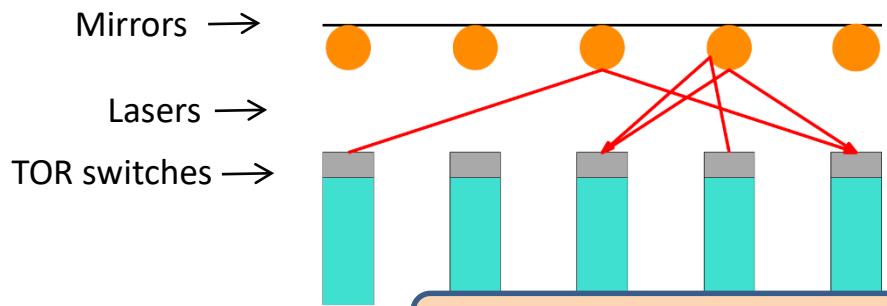
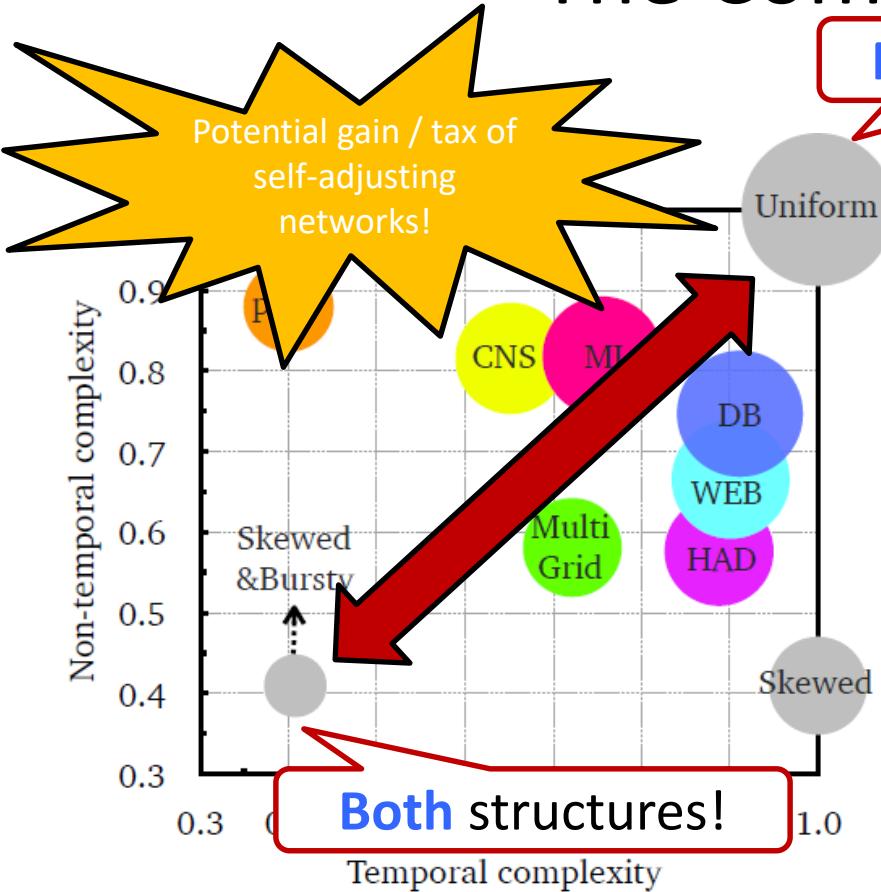
The Complexity Map



Goal: Design **self-adjusting networks** which leverage **both** dimensions of structure!



The Complexity Map



Measuring the Complexity of Packet Traces.
Avin, Ghobadi, Griner, Schmid. ArXiv 2019.

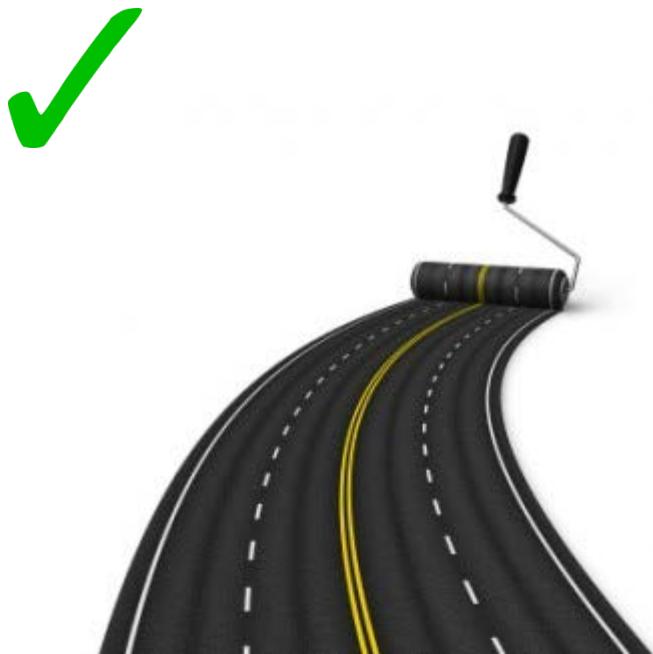
But: How to design DANs which
also leverage *temporal structure*?



Inspiration from **self-adjusting
datastructures** again!

Roadmap

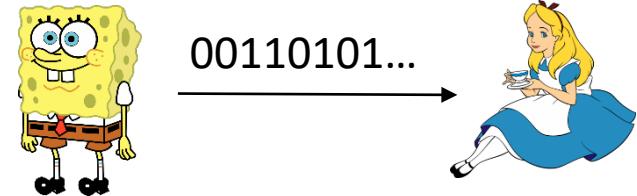
- Entropy: A metric for demand-aware networks?
 - Empirical motivation
 - A lower bound
 - Algorithms achieving entropy bounds
- From static to dynamic demand-aware networks
 - A connection to self-adjusting datastructures



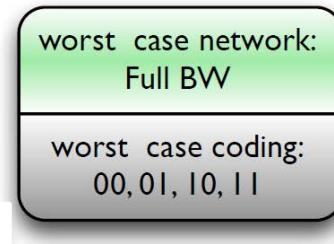
First: An Analogy

Static vs dynamic demand-
aware networks!?
DANs vs SANs?

An Analogy to Coding



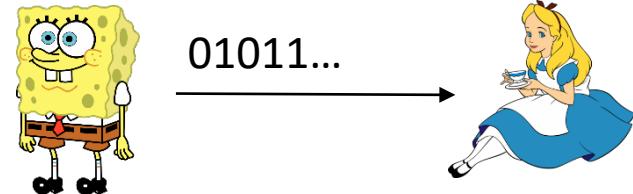
if demand **arbitrary** and **unknown**



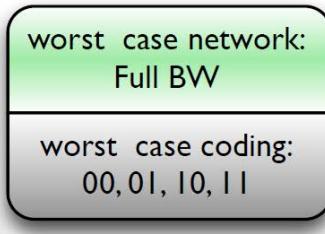
log diameter

log # bits / symbol

An Analogy to Coding



if demand **arbitrary** and **unknown**

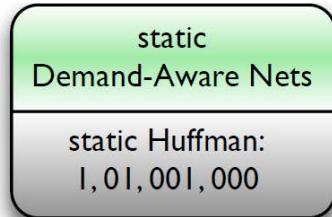


log diameter

log # bits / symbol



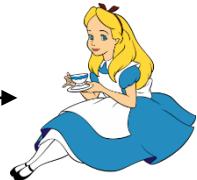
if demand **known** and **fixed**



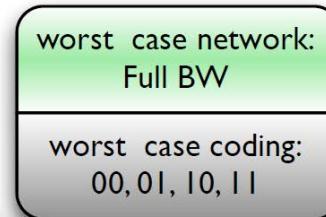
An Analogy to Coding



011...



if demand **arbitrary** and **unknown**



log diameter

log # bits / symbol

Dynamic DANs:
Aka. **Self-Adjusting Networks (SANs)!**

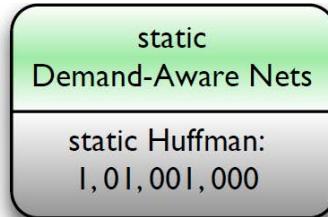


if demand **known** and **fixed**

if demand **unknown** but **reconfigurable**

entropy?

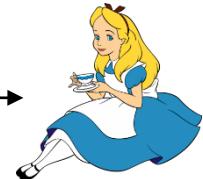
entropy / symbol



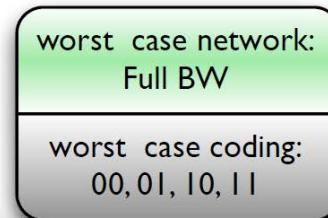
An Analogy to Coding



011...



if demand **arbitrary** and **unknown**



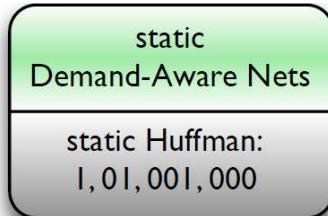
log diameter

log # bits / symbol

Dynamic DANs:
Aka. **Self-Adjusting Networks (SANs)!**



if demand **known** and **fixed**



Can exploit
spatial locality!

if demand **unknown** but **reconfigurable**



Additionally exploit
temporal locality!

An Analogy to Coding



011...



if demand **arbitrary** and **unknown**

worst case network:
Full BW

worst case coding:
00, 01, 10, 11

log diameter

Dynamic DANs:
Aka. **Self-Adjusting Networks** (SANs)!



if demand **know**



„Cheating“: need to know demand!



Can exploit
spatial locality!

Aware Nets
static Huffman:
1, 01, 001, 000

log # bits / symbol



if demand **unknown** but **repeating**

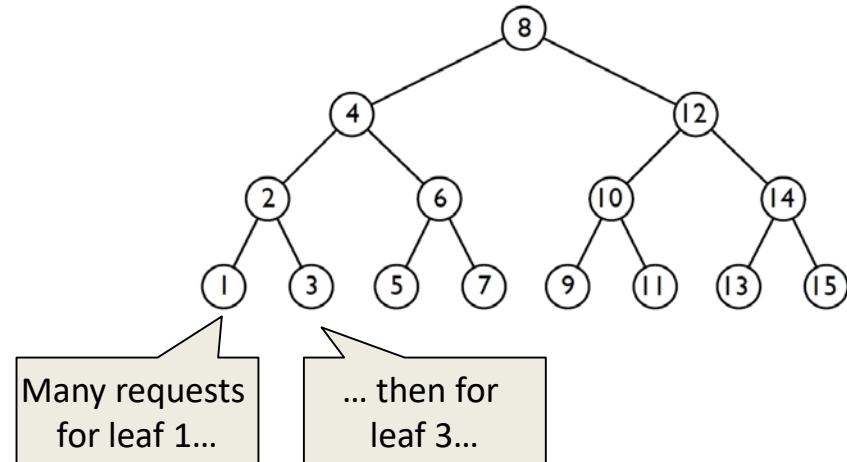
dynamic
Demand
Huffman codes

Need online algorithms!



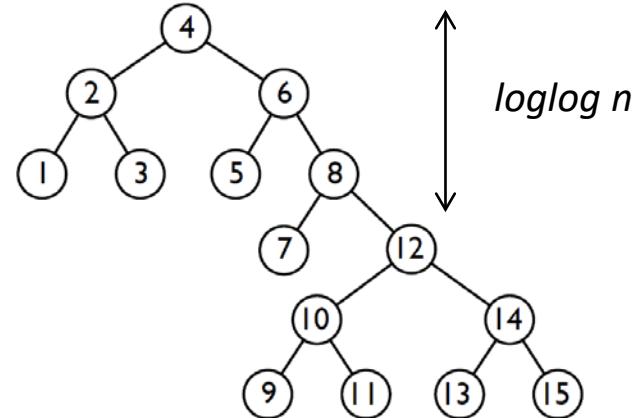
Additionally exploit
temporal locality!

Analogous to *Datastructures*: Oblivious...



... Demand-Aware ...

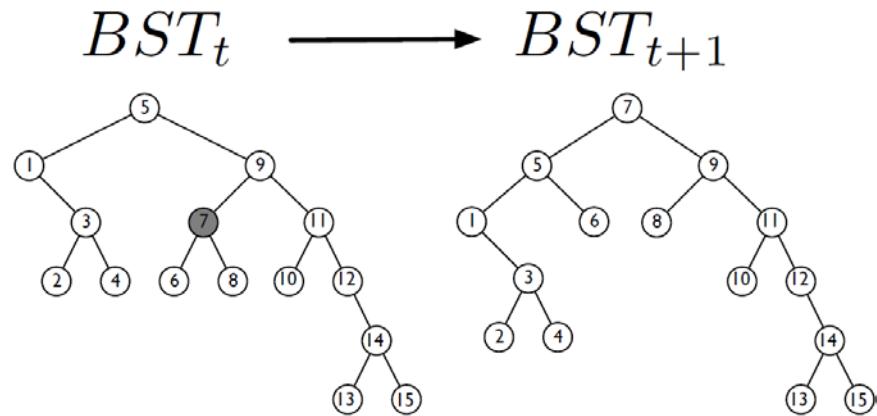
- **Demand-aware fixed** BSTs can take advantage of *spatial locality* of the demand
- E.g.: place frequently accessed elements close to the root
- E.g., **Knuth/Mehlhorn/Tarjan** trees
- Recall example **demand**:
 $1, \dots, 1, 3, \dots, 3, 5, \dots, 5, 7, \dots, 7, \dots, \log(n), \dots, \log(n)$
 - Amortized cost $O(\log \log n)$



 Amortized cost corresponds
to *empirical entropy of demand!*

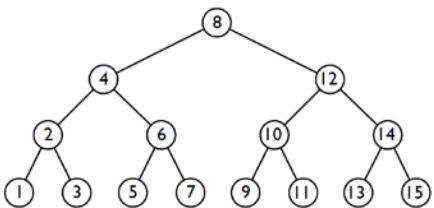
... Self-Adjusting!

- Demand-aware reconfigurable BSTs can additionally take advantage of *temporal locality*
- By moving accessed element to the root: amortized cost is *constant*, i.e., $O(1)$
 - Recall example demand:
 $1, \dots, 1, 3, \dots, 3, 5, \dots, 5, 7, \dots, 7, \dots, \log(n), \dots, \log(n)$

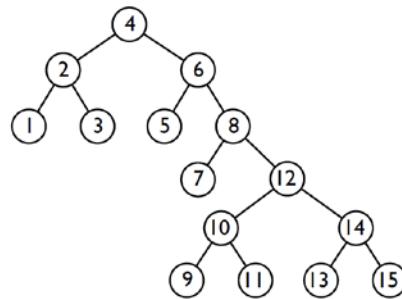


Datastructures

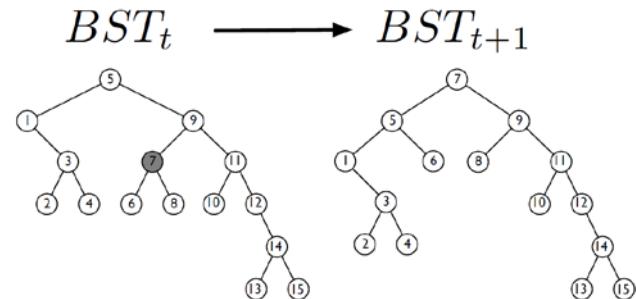
Oblivious



Demand-Aware



Self-Adjusting



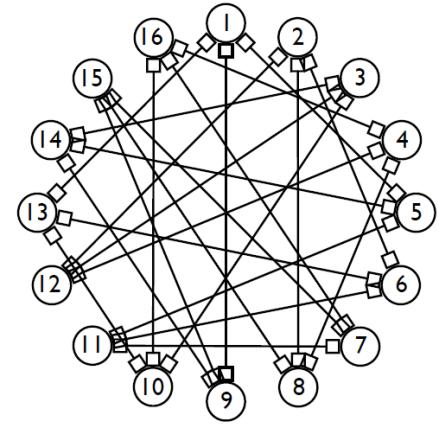
Lookup
 $O(\log n)$

Exploit **spatial locality**:
empirical entropy $O(\log\log n)$

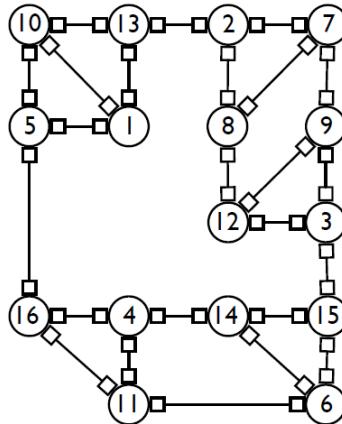
Exploit **temporal locality** as well:
 $O(1)$

Analogously for Networks

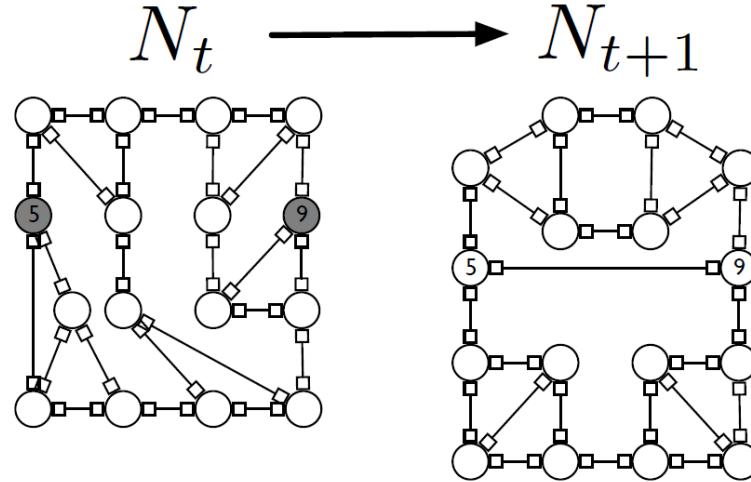
Oblivious



DAN



SAN



Const degree
(e.g., **expander**):
route lengths **$O(\log n)$**

Exploit **spatial locality**

Exploit **temporal locality** as well

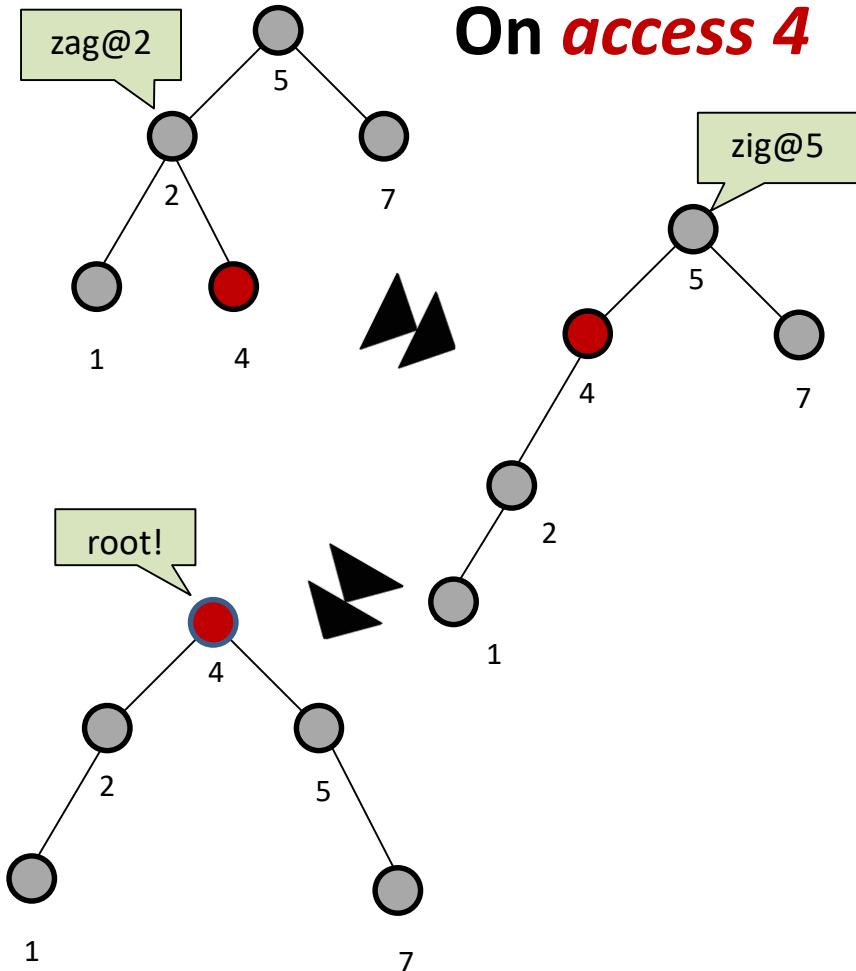
Now: Design of Self-Adjusting Networks (SANs)



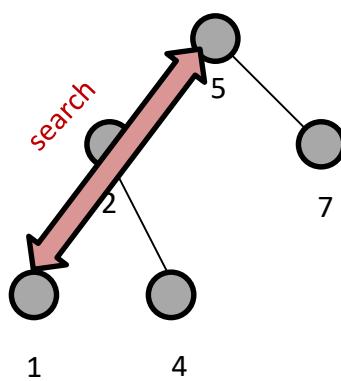
Inspiration from **self-adjusting
datastructures** again!

Recall: Splay Tree

- A Binary Search Tree (**BST**)
- Inspired by “**move-to-front**”: move **to root!**
- Self-adjustment: **zig**, **zigzag**, **zigzag**
 - Maintains **search property**
- Many nice properties
 - **Static optimality**, **working set**, (static,dynamic) **fingers**, ...

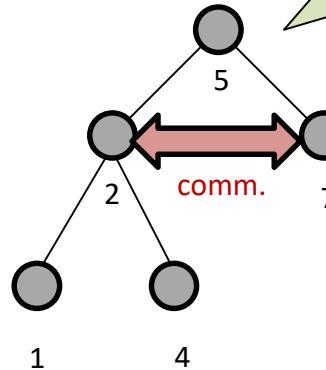


A Simple Idea: Generalize Splay Tree To *SplayNet*



Splay Tree

vs

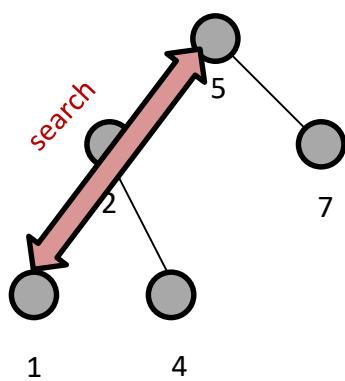


SplayNet

BST is nice for networks:
local (greedy) search!

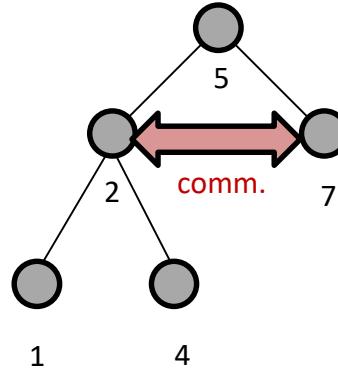
A Simple Idea: Generalize Splay Tree To *SplayNet*

But how?



Splay Tree

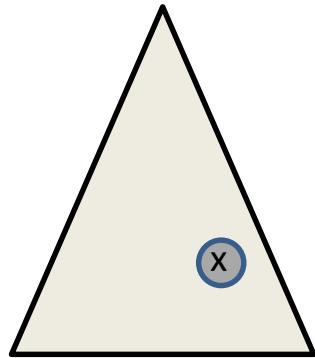
vs



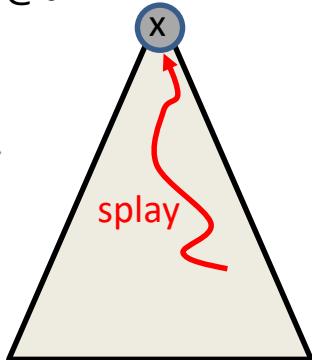
SplayNet

SplayNet: A Simple Idea

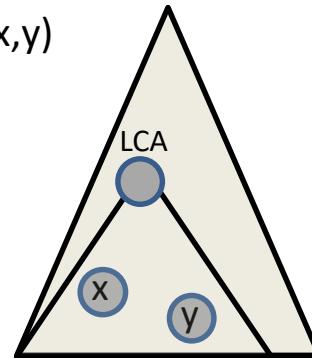
@t: access x



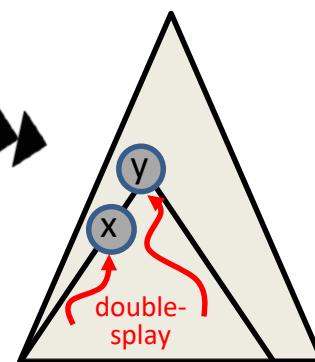
@t+1



@t: comm
(x,y)



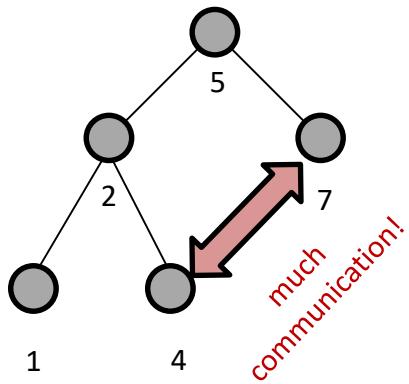
@t+1



Splay Tree

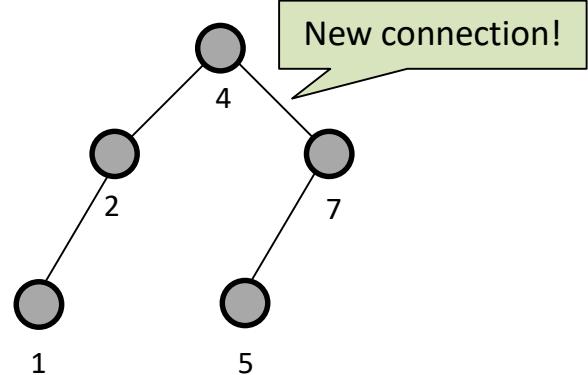
SplayNet

Example



$t=1$

adjust

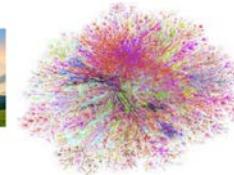


$t=2$

Challenges: How to minimize reconfigurations?

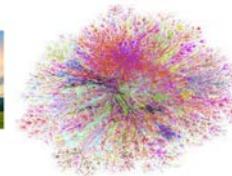
How to keep network locally routable?

Algorithms for Self-Adjusting Networks II



From trees to networks!

Algorithms for Self-Adjusting Networks II

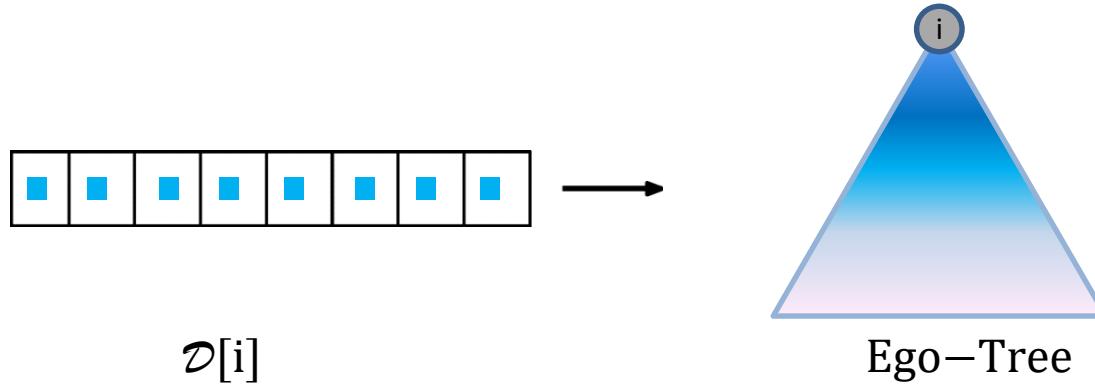


From trees to networks!

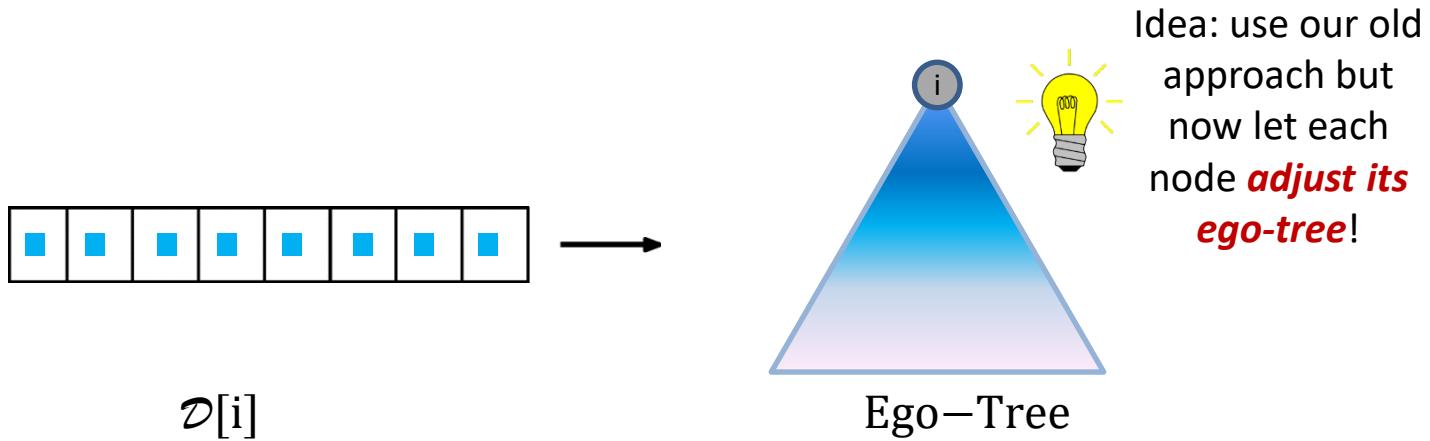


Ego-trees come-back!

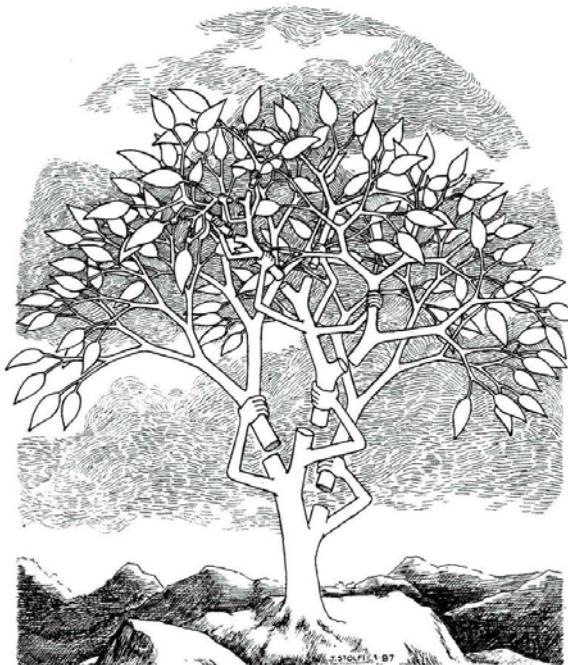
Total Recall: Ego-Trees!



Total Recall: Ego-Trees!



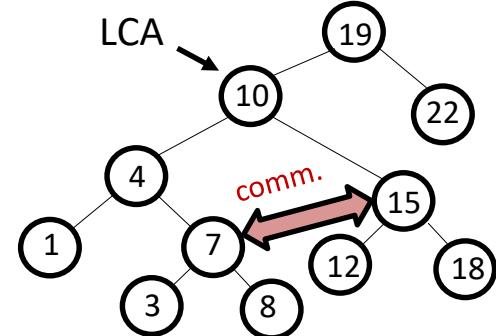
A Dynamic Ego-Tree: Splay Tree



Remark 1: Decentralized Algorithms

A “Simple” Decentralized Solution: Distributed SplayNet (*DiSplayNet*)

- SplayNet attractive: ordered BST supports **local routing**
 - Nodes **maintain three ranges**: interval of left subtree, right subtree, upward
- If communicate (frequently): **double-splay** toward LCA
- Challenge: **concurrency**!
 - But can be done: **independent clusters** moving towards LCA
 - Access Lemma of splay trees no longer works: **potential function** does not „**telescope**“ anymore: a concurrently rising node may push down another rising node again



SplayNet

Remark 2: Accounting for Congestion

A Tradeoff?!



Short routes:
congestion

VS

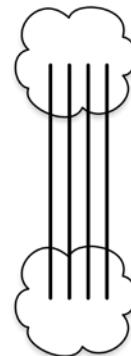


Low congestion:
long routes

A Tradeoff?!



Short routes:
congestion



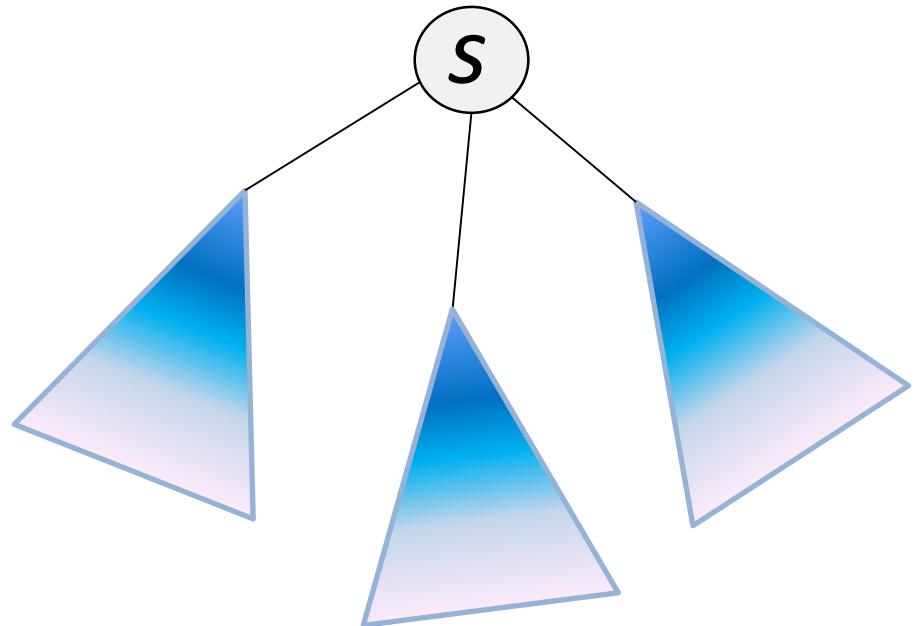
Or both?



Low congestion:
long routes

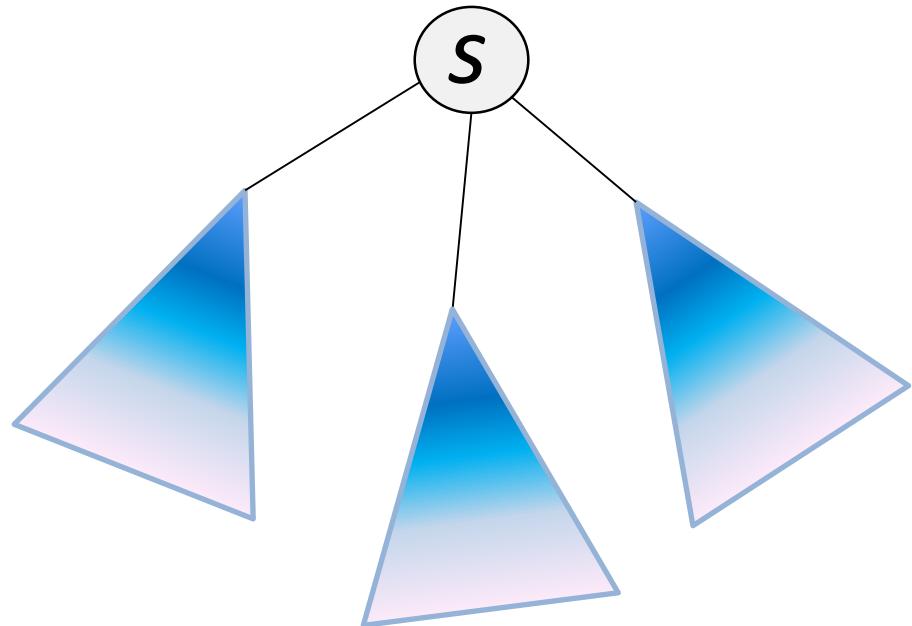
Ego-Tree++!

- Recall: ego-trees already are low-distortion / preserve distances
- Idea: additionally, place destination nodes greedily across subtrees s.t. ***congestion balanced***
- Trees can have different sizes but ***similar mass!***
- Bicriteria guarantee



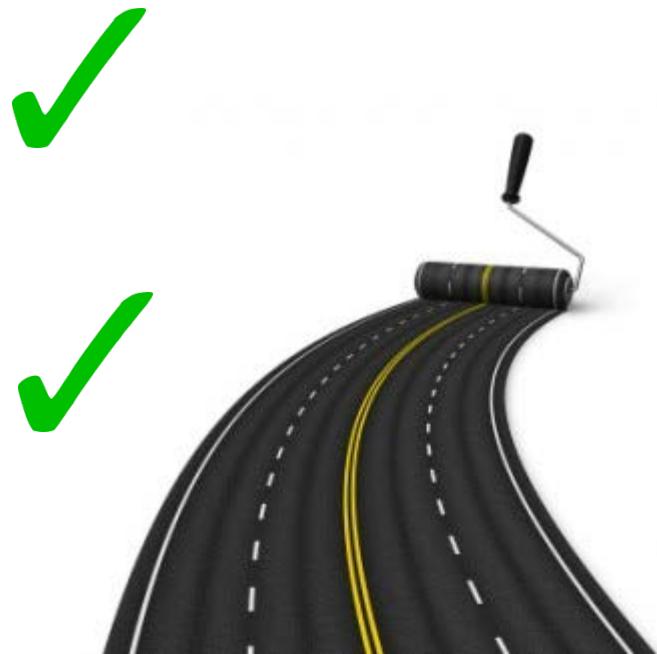
Ego-Tree++!

- Recall: ego-trees already are low-distortion / preserve distances
- Idea: additionally, place destination nodes greedily across subtrees s.t. ***congestion balanced***
- Trees can have different sizes but ***similar mass!***
- Bicriteria guarantee



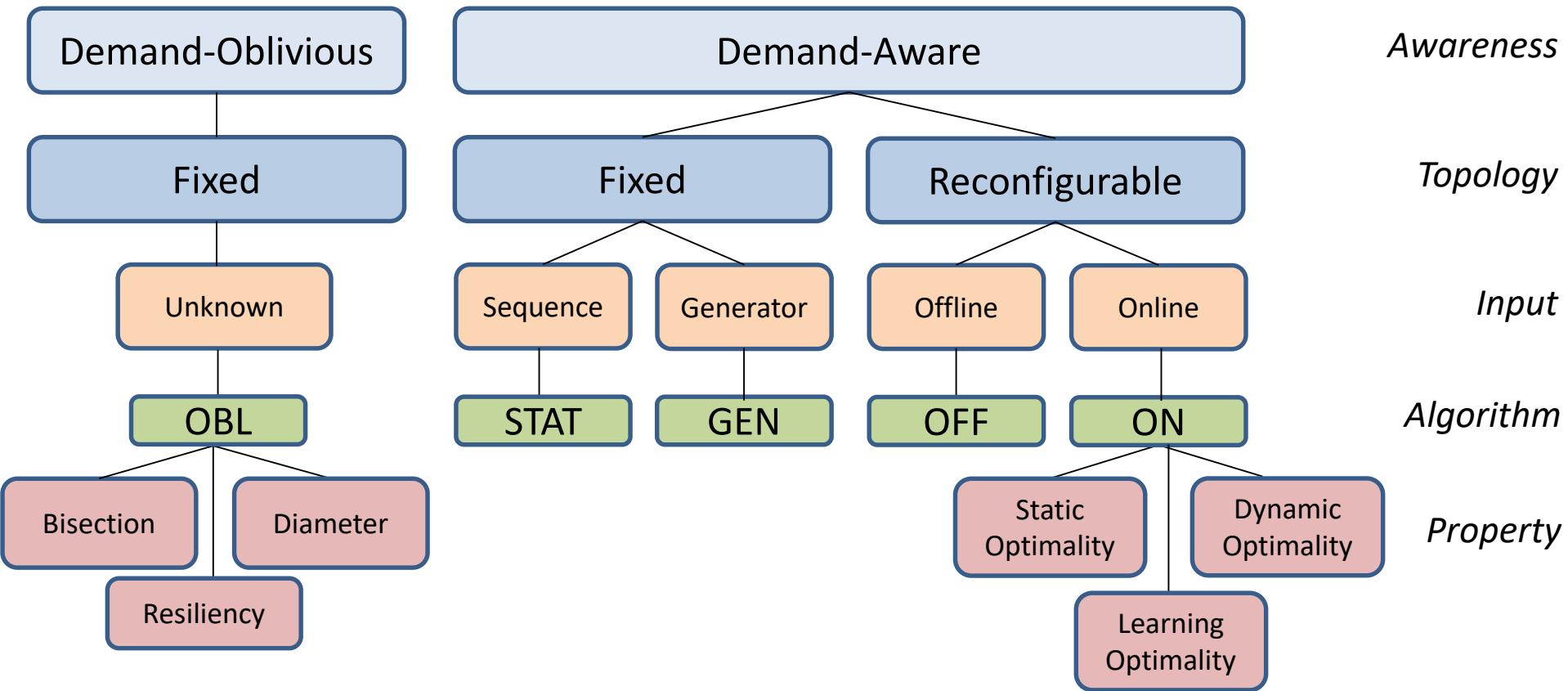
Roadmap

- Entropy: A metric for demand-aware networks?
 - Intuition
 - A lower bound
 - Algorithms achieving entropy bounds
- From static to dynamic demand-aware networks
 - Empirical motivation
 - A connection to self-adjusting datastructures



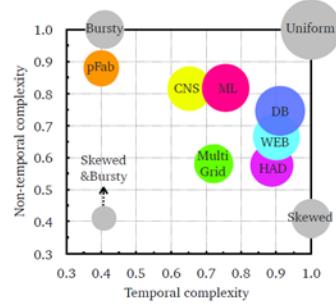
Uncharted Landscape!

Toward Demand-Aware Networking: A Theory for
Self-Adjusting Networks. SIGCOMM CCR, 2018.

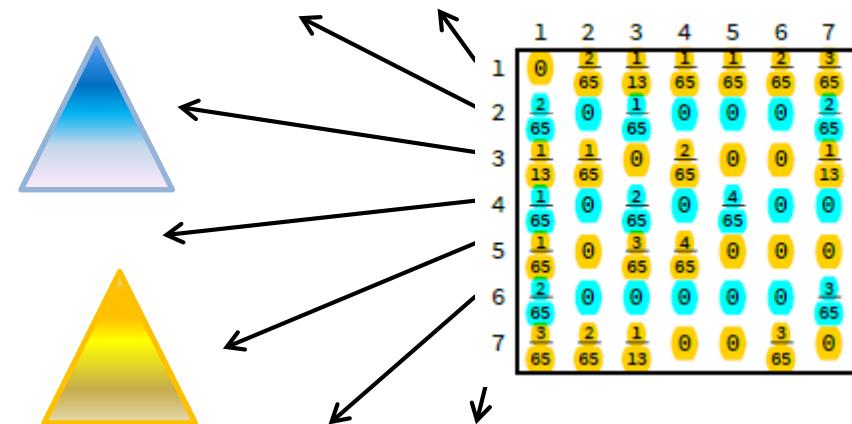


Open Questions

- Optimal static and dynamic topologies? Approximation algorithms?
- Scalable control plane?
- Cross-layer aspects
- Metrics: just the beginning!
- We need more data: how structured and predictable are workloads?
- How to convince operators?



Thank you! Questions?



Further Reading

[Survey of Reconfigurable Data Center Networks: Enablers, Algorithms, Complexity](#)

Klaus-Tycho Foerster and Stefan Schmid.

SIGACT News, June 2019.

[Toward Demand-Aware Networking: A Theory for Self-Adjusting Networks](#) (Editorial)

Chen Avin and Stefan Schmid.

ACM SIGCOMM Computer Communication Review (**CCR**), October 2018.

[Measuring the Complexity of Network Traffic Traces](#)

Chen Griner, Chen Avin, Manya Ghobadi, and Stefan Schmid.

arXiv, 2019.

[Demand-Aware Network Design with Minimal Congestion and Route Lengths](#)

Chen Avin, Kaushik Mondal, and Stefan Schmid.

38th IEEE Conference on Computer Communications (**INFOCOM**), Paris, France, April 2019.

[Distributed Self-Adjusting Tree Networks](#)

Bruna Peres, Otavio Augusto de Oliveira Souza, Olga Goussevskaia, Chen Avin, and Stefan Schmid.

38th IEEE Conference on Computer Communications (**INFOCOM**), Paris, France, April 2019.

[Efficient Non-Segregated Routing for Reconfigurable Demand-Aware Networks](#)

Thomas Fenz, Klaus-Tycho Foerster, Stefan Schmid, and Anaïs Villedieu.

IFIP Networking, Warsaw, Poland, May 2019.

[DaRTree: Deadline-Aware Multicast Transfers in Reconfigurable Wide-Area Networks](#)

Long Luo, Klaus-Tycho Foerster, Stefan Schmid, and Hongfang Yu.

IEEE/ACM International Symposium on Quality of Service (**IWQoS**), Phoenix, Arizona, USA, June 2019.

[Demand-Aware Network Designs of Bounded Degree](#)

Chen Avin, Kaushik Mondal, and Stefan Schmid.

31st International Symposium on Distributed Computing (**DISC**), Vienna, Austria, October 2017.

[SplayNet: Towards Locally Self-Adjusting Networks](#)

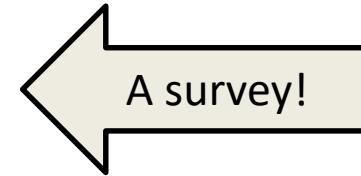
Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker.

IEEE/ACM Transactions on Networking (**TON**), Volume 24, Issue 3, 2016. Early version: IEEE **IPDPS** 2013.

[Characterizing the Algorithmic Complexity of Reconfigurable Data Center Architectures](#)

Klaus-Tycho Foerster, Monia Ghobadi, and Stefan Schmid.

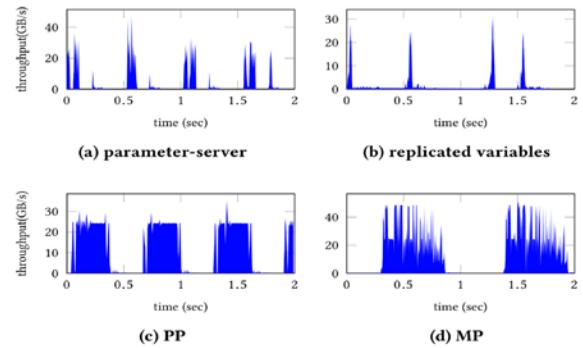
ACM/IEEE Symposium on Architectures for Networking and Communications Systems (**ANCS**), Ithaca, New York, USA, July 2018.



How Predictable is Traffic?

Even if reconfiguration fast, control plane (e.g., data collection) can become a bottleneck. However, many good examples:

- Machine learning applications
- Trend to disaggregation (specialized racks)
- Datacenter communication dominated by elephant flows
- Etc.



ML workload (GPU to GPU):
deep convolutional neural network
Predictable from their dataflow graph